University of Huddersfield Repository

Jidkov, Anton

A wireless real-time controller for the Apollo 'Ensemble' audio visual system

Original Citation

Jidkov, Anton (2013) A wireless real-time controller for the Apollo 'Ensemble' audio visual system. Masters thesis, University of Huddersfield.

This version is available at http://eprints.hud.ac.uk/18102/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

A WIRELESS REAL-TIME CONTROLLER FOR THE APOLLO 'ENSEMBLE' AUDIO VISUAL SYSTEM

ANTON JIDKOV

A THESIS SUBMITTED TO THE UNIVERSITY OF HUDDERSFIELD IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE BY RESEARCH

THE UNIVERSITY OF HUDDERSFIELD

JANUARY 2013

ABSTRACT

This research investigates the development of a surround sound controller prototype application for the Apple iPhone and iPad (iOS) for the users of Apollo Creative's 'Ensemble' system. 'Ensemble' is an interactive audio, lighting and video software and hardware package. It allows users to design multi-sensory environments and software instruments reacting to a range of input sensors. The users are primarily music therapists and special needs specialists. The iOS application functionality and user interface is designed primarily to accommodate Ensemble users but the study additionally explores complex functionality for advanced users such as composers and sound designers. Some of the features of the application allow users to: position multiple speakers arbitrarily in a 2D environment; adjust the volume of a single or groups of speakers; route sound source inputs; directly position sound; automate positioning by drawing pan trajectories; play back and manipulate the pan trajectories in a complex way including the use of touch gestures; further automate using multiple path layers. The software incorporates the OpenFrameworks library and communicates wirelessly with a Max/MSP spatialiser, using the Open Sound Control (OSC) protocol via control messages. Surround sound panning is controlled using Distance-Based Amplitude Panning to allow arbitrary speaker and listener positions. The application was evaluated by a series of diagnostic tests and detailed interviews.

CONTENTS

Section 1: Background	13
1.1 Introduction	14
1.2 Music Therapy	15
1.2.1 History	15
1.2.2 Practice	15
1.2.3 Models	16
1.2.4 Technology Use	17
1.2.4.1 Control Protocols: MIDI	18
1.2.4.2 Control Protocols: Open Sound Control (OSC)	18
1.2.5 Notation and Analysis Systems	19
1.2.6 Audio-Visual Systems and Controllers	20
1.3 Surround Sound	21
1.3.1 Introduction	21
1.3.2 History	23
1.3.3 Controllers	24
1.3.4 Panning and Spatialisation Techniques	26
1.3.4.1Pairwise Amplitude Panning and the Power Law	27
1.3.4.2 Vector Based Amplitude Panning	28
1.3.4.3 Ambisonics	28
1.3.4.4 Distance Based Amplitude Panning	28
1.3.5 Multichannel Live Sound	29
1.4 Touchscreens and Tablets	30
1.4.1 iPhone and iPad	30
1.4.1.1 Application Controllers	31
1.5 Apollo Creative Ensemble	33
1.5.1 Ensemble Hub	33
1.5.2 Input Sensors and Switches	34
1.5.3 Designer Software	35
1.5.4 Player	36
1.5.5 Devices Monitor	36
Section 2: Development	37
2.1 Users and Design Concepts	38
2.1.1 Primary Users	38
2.1.2 Advanced Users	40

2.2 System Specification	41
2.3 System Overview	42
2.3.1 Development Environment	42
2.3.2 Provisioning	43
2.3.3 iOS Integration	43
2.4 Application Design	45
2.4.1 Menu System	45
2.4.2 File Arrangement in Xcode	46
2.4.3 Graphical User Interface Setup	47
2.4.4 GUI Object Linking	47
2.4.5 Startup	47
2.4.6 New/Load Screen	48
2.4.6.1 New	49
2.4.6.2 Load	49
2.4.6.3 About	50
2.4.7 Open Sound Control Setup	50
2.4.7.1 Configuration Menu	52
2.4.8 The Move Menu	54
2.4.8.1 Reset Button	56
2.4.8.2 Grid Button	56
2.4.8.3 ID Pairing	58
2.4.8.4 Moving	59
2.4.8.5 Preventing Overlap	60
2.4.8.6 Window Translate	61
2.4.9 The Stepper	62
2.4.10 Volume Menu	63
2.4.11 Display Menu	65
2.4.12 Live Mode	67
2.4.13 Draw Mode	69
2.4.13.1 Drawing	69
2.4.13.2 Playing	71
2.4.13.3 Pausing	72
2.4.13.4 Reverse	72
2.4.13.5 Single Play	73
2.4.13.6 Speed Change	73

2.4.14 Touch Gestures	74
2.4.14.1 Function: 'AJpinch()'	74
2.4.14.2 Function: 'AJtouchgestures()'	75
2.4.14.3 Function: 'AJ3fingerrotate()'	77
2.4.15 Multiple Layers	78
2.4.15.1 Function: 'AJpathrecord()'	80
2.4.15.2 Playing: Function: 'AJpanlayerxplus()'	80
2.4.15.3 Pausing	83
2.4.15.4 Reverse	83
2.4.15.5 Single Play	84
2.4.15.6 Speed Manipulation	84
2.4.15.7 Touch Gestures	84
2.4.15.8 Layer 1 Time Display	85
2.4.16 Timing	86
2.4.17 Options Menu	86
2.4.18 Saving and Loading	88
2.4.18.1 Saving	89
2.4.18.2 Loading	90
2.4.19 OSC Message Sending Functions	92
2.4.19.1 Volume	92
2.4.19.2 Gain	93
2.4.19.3 Mute	94
Section 3: Evaluation and Discussion	95
3.1 Testing Tools and Techniques	96
3.1.1 Max/MSP Spatialiser	96
3.1.1.1 Configuration Messages	97
3.1.1.2 Master Volume	98
3.1.1.3 Source Routing and Sound Positioning	98
3.1.1.4 Sound Sources	99
3.1.2 Testing Environment	100
3.1.3 Diagnostics – Methodology	100
3.1.3.1 Activity Monitor: Total Load	100
3.1.3.2 Application Start	102
3.1.3.3 Speaker Movement	102
3.1.3.4 Path Playback	102

3.1.3.5 Touch Gestures	103
3.1.3.6 Additional Features	103
3.1.3.7 Message Sending Timing Accuracy	103
3.2 Discussion and Analysis	105
3.2.1 Application Start	105
3.2.1.1 Testing Results	105
3.2.1.2 Interviews	106
3.2.2 Setup: Speaker Movement	106
3.2.2.1 Testing Results	106
3.2.2.2 Interviews	107
3.2.3 Volume	108
3.2.3.1 Testing Results	108
3.2.3.2 Interviews	108
3.2.4 Paths: Live Mode	109
3.2.4.1 Testing Results	109
3.2.4.2 Interviews	110
3.2.5 Paths: Drawing and Playback	110
3.2.5.1 Testing Results: Total Load	110
3.2.5.2 Testing Results: Message Sending	112
3.2.5.3 Testing Results: Manipulation	114
3.2.5.4 Interviews	116
3.2.6 Saving/Loading	118
3.2.6.1 Testing Results	118
3.2.6.2 Interview	119
3.2.7 Fulfillment of Specification	119
3.2.8 Ensemble Integration	121
3.2.9 Future Development	122
3.3 Conclusion	124
Section 4: References	125
Section 5: Appendix	131

Word Count: 24427

FIGURE, TABLE, EQUATION LIST

Figure		
1.1	Flowchart representing the process of applying technology in practice (Magee & Burland, 2009)	21
1.2	Omnidirectional source radiates spherically. Sound energy passed through $1m^2$ of the surface at distance r will have expanded to cover $4m^2$ at a distance of 2r, giving rise to one quarter of the intensity or a 6dB drop. (Rumsey, 2001, p3)	22
1.3	Pyramix Surround Panner	24
1.4	Pro Tools Surround Panner	25
1.5	Cubase Surround Panner	25
1.6	Logic Pro Surround Panner	26
1.7	A Multisensory Room	33
1.8	The Ensemble Hub	33
1.9	The Dice Sensor in use	34
1.10	The Press Sensor	34
1.11	The Designer Software	35
1.12	The Player	36
2.1	The OF Code Structure	43
2.2	'ofxiPhone' operation	44
2.3	Application Menu Structure	46
2.4	Application Folders and Files	46
2.5	The New/Load Screen	49
2.6	The About Screen	50
2.7	Raw Messages	51
2.8	Configuration	52
2.9	Move Menu (no toolbars)	54
2.10	Move Menu (with toolbars)	55
2.11	Move Menu (with gridlines)	57
2.12	iPhone coordinate orientation	58
2.13	Volume mode with right speakers selected	63
2.14	The Display Menu	66
2.15	Live Mode	67
2.16	Draw Mode Buttons	69
2.17	Draw/Speed/Touch Button	69
2.18	Trajectory A	70

2.19	Snapshots of trajectory A playing on 2 layers	78
2.20	The combined movement of the sound for trajectory A playing on 2 layers	79
2.21	Time display markers showing a speed change	85
2.22	The Options Menu	87
2.23	The Save Menu	88
2.24	The Load Screen	91
3.1	Spatialiser - Presentation	97
3.2	Spatialiser – Volume Control and DAC patch segment	98
3.3	Spatialiser – 3 input panning patch segment	99
3.4	Spatialiser – Sound source selection	99
3.5	LMP Testing	100
3.6	Application startup trace	101
3.7	Spatialiser – message timing patch segment and cputimer subpatch	104
3.8	Speaker movement – CPU loading	107
3.9	Volume – CPU loading	108
3.10	Live mode – CPU loading	109
3.11	Path Playback – CPU loading	111
3.12	Message timing – Small Path	112
3.13	Message timing – Large Path	113
3.14	Moving a path – CPU loading	114
3.15	Resizing a path – CPU loading	115
3.16	Rotating a path – CPU loading	116
3.17	4-Speaker Example	121
3.18	Proposed Ensemble block architecture	122

Table		
1.1	Static Button Controllers	31
1.2	Customizable controllers	32
3.1	Application start – loading	106
3.2	Speaker movement – questionnaire	107
3.3	Volume - questionnaire	108
3.4	Live mode - questionnaire	110
3.5	Drawing a path – CPU loading	110

3.6	Single and Reverse button – CPU loading	114
3.7	Drawing a path – questionnaire	116
3.8	Path display – questionnaire	117
3.9	Multiple layers – questionnaire	117
3.10	Single/Reverse buttons – questionnaire	117
3.11	Touch gestures – questionnaire	118
3.12	Saving – CPU loading	118
3.13	File loading – CPU loading	119
3.14	Saving/Loading – questionnaire	119

Equation		
1.1	DBAP – Gains within the system are assumed to be normalized with constant energy. g is the gain at speaker, n, for N total speakers.	29
1.2	DBAP - the distance, d_i, from source to speaker for a total of N speakers.	29
1.3	DBAP - the value of the gain at speaker n; where d_n is the distance between source and speaker 'n'	29
1.4	DBAP – Adjustment for spatial blurring. $r_s>0$ is introduced into equation1.3.	29
2.1	The tangent rule	77

DIGITAL MATERIALS LIST

Max/MSP Spatialiser	The spatialiser used for testing. For Windows (.exe), Mac OS (.app) and the Max/MSP patch.
Video Demonstration	Video demonstrating the features of the study application (.mov).
Application Source Code	The study application code. Requires Xcode and Openframeworks (OF) libraries found at: <www.openframeworks.cc/download/>
MSc Thesis - Anton Jidkov U0770716 - January 2013	Electronic version of the thesis (.pdf).

GLOSSARY

App	Short for iOS application.
bang	A Max/MSP message that causes objects to trigger their output.
Bool	Short for 'Boolean'. A logical data type which has two values (true/false or 1/0).
buffer~	A Max/MSP object that stores audio samples.
cpuclock	A Max/MSP object that outputs the CPU time.
cycle	A Max/MSP sinusoidal oscillator.
dac	A Max/MSP digital to audio converter.
Int	Short for 'Integer'. A real, natural number without a decimal component.
ITU Standard	Following the recommendations of the International Telecommunication Union (ITU). E.g. for 5.1 surround sound.
Float	Short for 'Floating Point'. Use to represent numbers with decimal points.
metro	A Max/MSP object that acts as a metronome outputting a bang at a given interval.
Patch	A Max/MSP program.
play~	A Max/MSP object that plays a sample from a buffer~.
Pop	In programming, to remove an object from a stack.
Pointer	In programming, a data type that refers to the memory location of a variable.
prepend	A Max/MSP object that adds a message in front of input.
Push	In programming, to add an object from a stack.
route	A Max/MSP object that selects an output based on input matching.
sel	A Max/MSP object that outputs a bang based on input matching.
Stack	In programming, a last-in-first-out data structure.
Tapping	Touching a touch screen (on an iOS device).
udpreceive	A Max/MSP object that sends messages over a network.
udpsend	A Max/MSP object that receives messages over a netrowk
umenu	A Max/MSP drop down menu.
View	An iOS application is usually made up of 'views'. These can hold graphical user interface (GUI) components such as buttons, steppers.

When discussing the code variables, this study uses the forward slash (/) to signify multiple similar variables. E.g. The variable 'variableA/B/C' is referring to three variables: variableA, variableB and variableC.

ACKNOWLEDGMENTS

I would like to thank all those that contributed to this study. In particular I would like to thank Dr. Ian Gibson for providing guidance throughout the course of the study. Additionally, I am extremely grateful to Mark Hildred of Apollo Creative and to Tim Anderson for providing feedback. Finally I would like to thank the University of Huddersfield studio technicians for their support with the use of equipment and facilities.

COPYRIGHT STATEMENT

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Huddersfield the right to use such Copyright for any administrative, promotional, educational and/or teaching purposes.

ii. Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

iii. The ownership of any patents, designs, trade marks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions.

SECTION 1: BACKGROUND

1.1 INTRODUCTION

This project was embarked upon to research potential new application developments for Apollo Creative's Ensemble audio-visual system. The author had previously worked with adaptive technologies and music therapy in designing a music therapy software tool in Max/MSP. The tool executed a call and response game between therapist and client and provided analysis of the client response (Jidkov, 2011).

Apollo Creative is a company that designs and develops technologies aimed primarily at special needs learning. The Apollo Ensemble is a system designed for teachers and special needs specialists, allowing them to configure interactive sensory environments for individuals with a range of disabilities. Ensemble also has applications in the areas of exhibitions, artistic installations and children's play areas (Apollo Creative, n.d.).

Ensemble's benefits are that it is entirely plug and play and has a wide range of customized sensory equipment (ibid). It is designed to make it easier for non-technical users to setup and operate the system.

The study application is intended for use on iOS devices (iPhone and iPad). Student use and parental acceptance of this technology is growing due to increased ownership of the devices (Heinrich, 2012, p7). The fact that it is a fashionable item means that non-technical users are more likely to have an interest in developing skills rather than being discouraged by the new object. This applies to teachers as well as children (ibid, p46).

The application will serve as a proof-of-concept focusing on the surround sound implementation of the Ensemble system. It will allow the user to control the movement of sound. The application is aimed primarily at special needs teachers. Therefore the level of technical proficiency, and the issues this raises, must be taken into account. The application also aims to have more complex controls and functionality so that it could be used by a more technically proficient group including: composers, sound designers and installation artists.

1.2 MUSIC THERAPY

1.2.1 HISTORY

The use of music for therapy and healing has a history stretching back to 1800 BC where the oldest account of medical practices, the Kahum papyrus, describes the use of incantations for healing (Wigram, 1995, p9). In 1500 BC, Egyptian medical papyri state the power that music has on fertility (Jorda, 2008, p774). The Turco-Persian psychologist, Abū Naṣr al-Fārābī (circa 878BC) also details the use of music in a therapeutic way in the treatise 'Meanings of the Intellect' (Haque, 2004, p363).

However, it was not until the 18th Century when scientific investigations on how music affects the body began to take place. Notable examples include J. Bonnet (1715), Dr Chomet (1875) and L. Roger in (1748).

An important step of contemporary music therapy occurred during World War I and World War II. Harriet Ann Seymour used music to relive traumatic injuries and improve psychological well being of the soldiers, and went on to found the National Institution of Music Therapy in 1941 (AMTA, 2011). In the UK, Juliette Alvin founded the Society of Music Therapy and Remedial Music in 1958 and a training program in Guildhall School in 1967 (Aigen, 2005). Currently, music therapy practice is coordinated by the 'British Association for Music Therapy' and a range of degrees are now available which have led to a surge of therapy practitioners (BAMT, 2012).

1.2.2 PRACTICE

The practice of music therapy ranges significantly depending on a variety of factors: cognitive capacity, age and the presence of mental disorders. Bunt (1994, p6), describes it as:

> '[An attempt] *to establish a communication with another human being where music is the means by which this is achieved'.*

The practice is also described by Juliette Alvin (ibid, p6) as:

> *'The controlled use of music in the treatment, rehabilitation, education and training of children and adults suffering from physical, mental or emotional disorder'.*

The 'National Association for Music Therapy' (NAMT) (Davis et al, 1998, p5). describes it as:

> '*The use of music in the accomplishment of these therapeutic aims: the restoration, maintenance, and the improvement of mental and physical health. It is the systematic application of music, as directed by the music therapist in a therapeutic environment, to bring about desirable changes in behavior.*'

1.2.3 MODELS

Music therapy is used within a wide range of clinical practices. Top of the list for adults and children are learning difficulties followed by emotional, behavioral and mental difficulties (Bunt, 2002). Historically, this is due to the fact that the first trained music therapists began work in institutions dealing with these groups (Bunt, 1994, p9). Pioneering music therapy was found to be helpful with these groups and resulted in systematic research (ibid).

Music therapy models can be described as 'part of a continuum ranging from a directive to non-directive approach' (ibid, p140). The main models are described below.

- Helen Bonny's work on Guided Imagery in Music (GIM) which uses music as a trigger to guide the client through an inner journey (ibid, p190).
- The Priestly model – analytically orientated therapy; creative music therapy which is more relationship based and allows clients to develop self organization and experience in relating to others (ibid, p7/44).
- The Alvin model – free improvisation and behavioral music therapy (Oldfield, 2006).

Although therapists usually favor one method due to their clients and training, usually elements from several models are used on a case-by-case basis and many models have large overlapping areas. The shared aims of these models are to improve self-awareness – identity, confidence, ability and relationship to others and communication – attention, imitation and 'cause and effect'. In particular, 'cause and effect', the connection between actions and environment, is very important to develop (Corke, 2002).

1.2.4 TECHNOLOGY USE

The use of electronic music technologies can offer new and adaptable ways for a client to interact with a musical environment and novel methods have been shown to have a strong impact on outcomes relating to communication and expression (Magee, 2009).

A secondary aspect is that there is a growing pressure on therapists to evaluate practices in a quantifiable way. Due to the range of conditions, types of practices and approaches, there is no standardized music therapy system. Methodologies are affected by the client themselves, individual therapists, the ethos of the clinical setting and the original training of the therapist. Currently, strongly contrasting opinions exist stating either that standardization is essential or that a linear method is incompatible in the uncertain nature of the practice (Streeter, 2010, p52).

Technology has started to be incorporated into therapy for the past 20 years (Kirk et al, 2002). However this area of music therapy is still underdeveloped and has a lack of guidelines for incorporation of technologies. The benefits of music therapy technologies are clear with regards to certain groups. These include adolescents who have otherwise found it difficult to engage in therapy and clients with especially complex needs (ibid).

The use of technology in many ways provides a 'blank sheet of paper' in terms of sound design and control. More importantly it can 'decouple the nature of gestures…from the resulting sound' (ibid). With regards to sound creation, acoustic instruments require specific gestures and often an advanced skill level in order to manipulate them with satisfactory outcomes. Conversely, electronic instruments can allow small physical movements to control large and expansive sounds. These instruments can also be tailored to a person's specific mobility and skill level. This allows an unskilled client to play a complex scale or chord sequence on piano and guitar with a slight pressure variation of the finger or with a tilt of the head. Additionally, activity using electronic devices can be recorded, analyzed and, potentially, quantified.

There are of course, certain drawbacks to the standard electronic device. This includes lack of haptic and visual feedback and issues with tactility. Research is being done to focus on these aspects of electronic devices to make them more 'natural'. The EU project 'Care Here' use the phrase 'Aesthetic Resonance' to describe a state whereby

the feedback loop between an audio-visual system and client is so strong that the creation of movement is forgotten (Hunt et al, 2004).

1.2.4.1 CONTROL PROTOCOLS: MIDI

One of the important advances for technology use in this field was the introduction of the Music Instrument Digital Interface (MIDI) in 1983 (MIDI Manufacturers Association, 2012). MIDI is a digital communications language and specification. It allows hardware and software equipment, computers and controllers to communicate over a network (Huber, 2007). MIDI does not transmit or create audio but sends control messages that serve as on/off triggers for a wide range of parameters (e.g. note on/off, control change). This type of protocol is versatile and can be used to play a synthesizer, trigger sounds or clips on a computer and is useful in data analysis – see section 1.2.5.

Messages take the form of a status byte and 2 data bytes and are traditionally transmitted via a MIDI line (5 pin DIN) in series at a speed of 31,250 bits/sec (ibid). However transmission in different forms is now common and includes USB, Firewire and Ethernet, greatly increasing transmission speed. USB 2.0 can achieve 480Mbits/sec and Firewire 800 can achieve 800 mBits/Sec.

Currently, MIDI is the preferred protocol. However it does have limitations including a slow transfer speed and potential lag if many devices are present; a limited instruction set and a limited resolution of 127 (Huber, 2007).

1.2.4.2 CONTROL PROTOCOLS: OPEN SOUND CONTROL (OSC)

Open Sound Control (OSC) is an 'open, efficient, transport-independent, message-based' protocol used for communication between a variety of multimedia devices such as computers and synthesizers (Wright, 1997). It was originally developed at The Centre of New Music and Audio Technologies (CNMAT) in the University of California, Berkeley for transmitting music performance data between instruments and is often used in place of the 1983 MIDI standard. It provides an 'open-ended, dynamic naming scheme' which makes it appropriate to use in a variety of data sending applications. The format allows for future proofing (CNMAT, 2012).

OSC gives users more functionality compared to MIDI's limited addressing model and numerical precision. It provides a symbolic naming structure that is easier to use than the arbitrary mapping that MIDI provides in the form of channel, program change and controller numbers (Wright, 1997).

OSC's basic unit is the 'message'. It consists of

1. A symbolic address and message name.
2. A type tag string specifying the data type of each argument (e.g. floating point).
3. Any amount of binary data following the message in the form of arguments (Wright, 1997).

1.2.5 NOTION AND ANALYSIS SYSTEMS

Computers were being used since the 1980s to notate and evaluate music therapy data by collecting and recording simple interactions between therapist and client (Hasselbring and Duffus, 1981). Music therapists also used a range of notation tools that were not specifically designed for the practice including SCRIBE, AIMSTAR and EMTEK (Hahna et al, 2012).

Lee (2000) was the first to use computer notation software in the late 80s and early 90s to transcribe and analyse music therapy improvisations. This led to the development of a more advanced system, 'The Computer Aided Music Therapy Analysis System' (CAMTAS), in the mid 90s by Verity and Kirk (Hunt et al, 2000) to organize and compare data taken from audio and video recordings and was designed for use in improvisation environments over the course of several sessions. At the time, the computational power and the awkward system setup provided a range of limitations and forced therapists to change the way that sessions were run to accommodate (Streeter, 2010).

More contemporary system include:

- The Music Therapy Toolbox: Open source MIDI analysis software (Erkkilä, 2007, pp. 134-148).
- MAWii Music Therapy System: an open source system focusing on gathering data from Wii-motes used in group sessions (Benveniste et al, 2008).

- The Music-therapy Analyzing Partitura (MAP): a system allowing subjective, qualitative annotation using set event types (Gilboa, 2007, pp. 309-320).
- The Individual Music Therapy Assessment Profile (IMTAP): collects and manages data from assessments (Baxter et al, 2007).
- The Music Therapy Logbook: collects and analyses data from acoustic and MIDI recordings (Streeter, 2010).

1.2.6 Audio-Visual Systems and Controllers

Audio-visual systems such as Ensemble benefitted greatly with the introduction of MIDI. However it has limitations, as mentioned in section 1.2.4.1. Today, most of the systems on the market use MIDI:

- Midigrid: A software program that allows sound events to be triggered via mouse on screen (Hunt and Ross, 2003).
- Midicreator: A device, very similar to Ensemble, that allows a variety of contact sensors to be plugged into the device which are then converted to MIDI notes and chords (MIDIcreator, n.d).

Some packages are built around a particular sensor providing a software interface for control. These include:

- Skoog: a squeezable cube instrument sending MIDI commands to a software interface to control music output (Skoogmusic, 2012).
- Soundbeam: A contactless conversion from movement into music. Uses ultrasonic beams to send MIDI messages to a software interface. Limited to one software package and no haptic feedback (Soundbeam Project, 2012).

One of the main functions of these systems is to encourage the development of cause and effect as discussed in section 1.2.3. They convert instantaneous (direct) movement 'gestures' into sonic output. Small movements from an inexperienced player can be converted into coherent, well-structured and large-sounding output (see section 1.2.4). Sensors usually provide some haptic and visual feedback to the player with the notable exception of Soundbeam. However, Soundbeam is one of the most popular devices (Magee & Burland, 2009) potentially due to the fact that it allows a wide range of gestures to be created without slowing the process by changing sensors/wiring.

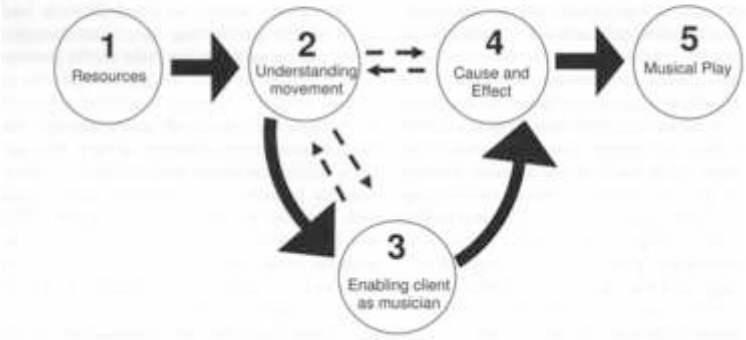


Figure 1.1 Flowchart representing the process of applying technology in practice (Magee & Burland, 2009)

Figure 1.1 shows the sequential phases of the clinical process when using the devices mentioned in this section. It highlights the need for understanding the client's requirements in terms of movement patterns and selecting and using appropriate technology. Only when 'cause and effect' is developed sufficiently, musical play can begin.

1.3 SURROUND SOUND

1.3.1 INTRODUCTION

The sonic world is composed of a complex soundscape arriving at a listener's ear in all three dimensions. Sound can be categorized by the brain in a variety of ways including its location, size and content.

Indoor environments tend to be characterized by reflections due to the enclosing surfaces. The length and phase of these reflections can help assessment of the size, as well as other qualities, of a room. Outdoor environments also experience reflections such as obstacles and reflections from the ground.

Spatialisation in sound can therefore be split into 'source' and 'environment' groups. Source sounds are direct and localizable and environment sounds are diffuse and ambient. In an environment with no reflections (free field) all sound is radiated away from the source. Omnidirectional sound is radiated spherically (dependent on frequency content) and results in a 6dB level drop, or a quartering of intensity for every doubling of distance as shown in figure 1.2.

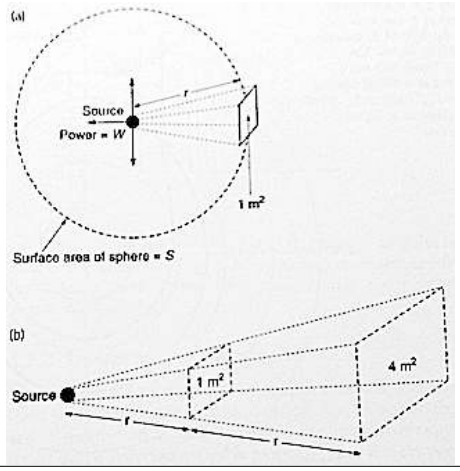


Figure 1.2 Omnidirectional source radiates spherically. Sound energy passed through $1m^2$ of the surface at distance r will have expanded to cover $4m^2$ at a distance of 2r, giving rise to one quarter of the intensity or a 6dB drop. (Rumsey, 2001, p3)

The localization of a sound occurs in the brain by 2 main processes: timing/phase and level/spectral differences between the ears. The importance of these depends on the nature of the sound and the accompanying reflections.

This interaural time difference (ITD) depends on the angle of incidence of the source and allows for source resolution of a few degrees with the maximum time difference being 0.65 ms when the sound is directly to the right or left of the listener. The brain tends to use this method for sound transients and primarily with low frequency (LF) content. Off-centre sources will also have an amplitude difference between the ears. This effect is usually most important with close sources. Also, the shape of the head, outer ear (pinna) as well as reflections off the torso will alter the spectral quality of the sound. These effects can be summated as the head related transfer function (HRTF).

The precedence effect occurs when a similar source arrives from 2 or more locations (when the time difference is < 50 ms). This effect is often used with loudspeakers where copies of the same sound, with time/amplitude differences, are played from several locations and the brain localizes depending on the arrival of signals (Rumsey, 2001).

The benefits of multichannel surround sound are that it enhances envelopment (being in the space of the recording or mix) as well as spaciousness (looking into a window containing 'space'). Additionally, surround sound allows separate sound elements in a mix to be separated so that they can be heard and localized more clearly. This is especially important for music therapy for several reasons. Firstly, in a situation with

multiple players contributing to a performance, it allows individual players to hear themselves clearly in the mix if enough separation between sources is available. The ideas of envelopment can help the client feel connected to a performance or part of a realistic sensory experience. These factors are very important in developing cause and effect.

1.3.2 HISTORY

The history of surround sound is complex and erratic. It is an area of music technology that is still rapidly changing and developing. The first demonstration of stereo sound occurred in the 1930s when audio was transmitted by Bell Labs from the Academy of Music in Philadelphia to Washington, DC and was reproduced on a 3 channel speaker system. Further developments by Disney, for 'Fantasia' 1938-41, lead to the development of a range of tools and techniques including a precursor to the current standard 5.1 systems, multi-track recording, pan potting and overdubbing.

The end of the war brought new developments including high quality loudspeakers and magnetic tape recording but also heralded a commercial shift away from people attending cinemas to home television. This allowed 2 channel stereo, a simplification of cinema systems as well an uncomplicated signal quantity in the medium of vinyl records, to become the dominant model until 1975.

Although home sound continued to be 2 channel, cinema sound developed and experimented with various techniques including quad – four signals decoded from two tracks using amplitude and phase relations. This particular development lead to Dolby Stereo. The way in which frequency content is distributed between speakers was developed and a low frequency channel was introduced as standard. This was formalized by the Society of Motion Picture and Television engineers to the standard of 5.1.

Superior home formats started developing such as VHS, which greatly improved signal-to-noise ratio, and Laser Disc, which introduced 2 digital 44.1kHZ, 16-bit linear PCM tracks into domestic sound. Following this, the home sound system has been allowed to expand to accommodate affordable multichannel formats (Holman, 2000).

1.3.3 CONTROLLERS

Panners for mixing consoles have historically come in 2 types: the 3-channel/knob panner: left-centre-right, front-back and left surround-right surround; the joystick control, which can be less precise in terms of determining position in the mix but allows easier movement. The joystick puts emphasis on the internal area of the speakers and has sound playing from multiple speakers simultaneously. This can cause problems in terms of localization as the listener may hear sounds arriving from the closest speaker. There are also problems relating to frequency content affected by head related transfer functions (HRTFs). Contemporary panners are designed for Digital Audio Workstations (DAWs). An important development with software-based controls is automation.

Surround sound control developments tend to be tailored towards the professional sector. This includes cinema post-production and, to a lesser extent, music studio production and mastering. Several DAWs and the available 'in-the-box' surround sound options are discussed. The discussion will focus on visual, joystick-type positioning implementations.

Pyramix (Merging Technologies) [v7]:

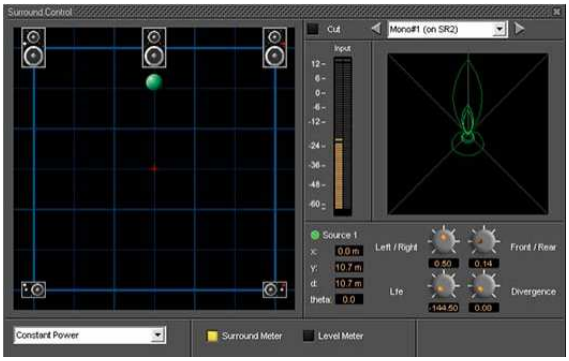


Figure 1.3 Pyramix Surround Panner

Figure 1.3 shows a 5.1 speaker setup. It allows the positioning of both mono and stereo (not shown) sources with 'full automation support'. Stereo sources can be controlled independently or linked in various ways (absolute, relative, mirror). A drop down menu allows for constant power or constant gain panning algorithms. An independent low frequency level control is available (Merging Technologies, 2011).

Protools (Digidesign, Avid) [v10 and HD]:

Figure 1.4 Pro Tools Surround Panner

Pro Tools HD and Pro Tools with Complete Production Toolkit allows standard surround formats up to 7.1. The panner is shown in figure 1.4. An X/Y grid is available for positioning mono and stereo sound sources. Stereo sources allow linking. The figure above also shows 3-knob positioning. An additional LFE level is provided. Automation is available. 'Autoglide' mode allows quick writing of surround automation by setting the destination and the time (Avid Technology, 2011).

Cubase (Steinberg) [v6]:

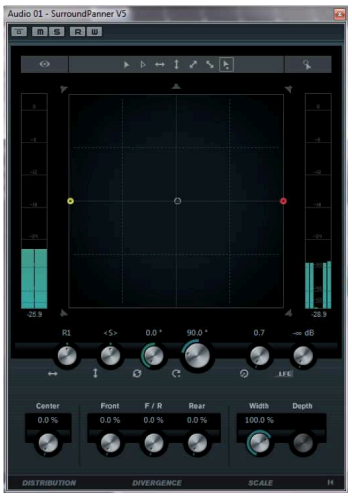


Figure 1.5 Cubase Surround Panner

The surround sound plug-in, SurroundPanner V5 is automatically applied to multi-channel tracks and covers the standard surround formats, shown in figure 1.5. This panner employs the constant power panning algorithm. Positioning mono and stereo sound involves clicking and dragging within the display window. The user can be aided by limiting movement (e.g. front/rear only) or using the rotation and orbit controls (Bachmann et al, 2010).

Logic Pro (Apple) [v9]:

Figure 1.6 Logic Pro Surround Panner

Logic Pro offers the standard surround formats for mono and stereo source panning and an independent LFE and centre channel control. Again, an X/Y grid is available, where the user can position the source. Logic pro also has the option for binaural panning (Apple, 2011).

Overview: Users of these products will have an idea of what they want the sound to be and will use other techniques (such as delay and phase) as well as the panner to add to spatialisation and depth perception. The speaker positions are very specific formats with the listener directly in the sweet spot. The Pro Tools manual explicitly states that '*It is very important that your surround monitor system be installed and configured correctly. Proper speaker placement, angling, and level calibration are necessities for surround mixing.'* (Sync & Surround Concepts Guide, Avid Technologies, 2011, p17).

1.3.4 PANNING AND SPATIALISATION TECHNIQUES

Panners perform a series of operations on the sound channels. Usually the operation of panners attenuates one channel while intensifying another. There is still debate as to the appropriate rules to employ and variation occurs between panners.

Gerzon (1992) describes panning laws as follows:

> *The aim of a good panpot law is to take monophonic sounds, and to give each one amplitude gains, one for each loudspeaker, dependent on the intended illusory directional localisation of that sound, such that the resulting reproduced sound provides a convincing and sharp phantom illusory image. Such a good panpot law should provide a smoothly continuous range of image directions for any direction between those of the two outermost loudspeakers, with no "bunching" of images close to any one direction or 'holes' in which the illusory imaging is very poor.*

Multichannel panning (>2 speakers) presents problems of energy distribution, localization, off-centre (outside of the 'sweet spot') listening, timbral changes (Rumsey, 2001) and the precedence effect (section 1.3.1). This section will briefly discuss several horizontal (2 dimensional) panning techniques.

1.3.4.1 PAIRWISE AMPLITUDE PANNING AND THE POWER LAW

A common approach is the pairwise amplitude panning technique. The technique uses changes in amplitude between 2 adjacent speakers to create a phantom image at a position between the speakers.

The 'Power law' is often used to determine the relative amplitudes of the speakers. This law is credited to Blumlein who, in 1931, developed it as a generalization of his theoretical psychoacoustic methods (Blumlein, 1933 and Gerzon, 1992). The attenuation relationship follows the sine/cosine laws with an appropriate crossover at the centre. Once again, the early research into this topic was carried out by Disney, which found that the power law is the most appropriate for panners. They also found that an attenuation of 3dB was appropriate for a sound halfway between two speakers (Holman, 2000).

The power law is a simplified approximation of actual psychoacoustics. This means it has some limitations including:

- Problems with wide speaker placement.
- Problems with off-centre listening.
- Side phantom image problems including:
    - Poor localization.

- Non-linear movement.
- Spectral problems including 'smearing'.

1.3.4.2 VECTOR BASED AMPLITUDE PANNING

Vector based amplitude panning (VBAP) is an extension of the amplitude panning model developed by Pulkki (1997). It is an amplitude panning method extended by using vectors, vector bases and adjacent speaker pairs which can be extended into 3 dimensions by using speaker triplets. The method allows for any number of loudspeaker with arbitrary placements.

However, it assumes a known and fixed listener position and is not intended for placing a source within a speaker array – only on the speaker arc.

1.3.4.3 AMBISONICS

Ambisonics was developed by Gerzon (1973). It aims to be a complete hierarchical approach to directional sound pickup, storage and reproduction and requires encoding and decoding of sources that are to be presented on a sound field. The basis of this technique was to improve localization based on the low and high frequency models of human hearing (Rumsey, 2001).

Once again, limitations of this model assumes a small 'sweet spot'; not intended for sources within the array; the speaker positions are limited, typically requiring the speaker to surround the listener in a sphere (Kostadinov et al, 2010).

1.3.4.4 DISTANCE BASED AMPLITUDE PANNING

Distance based amplitude panning (DBAP) was first developed by Trond Lossius (2009). It is an alternative panning spatialisation method that allows an arbitrary speaker setup and no listener position or 'sweet spot' (Lossius & Hogue, 2009). DBAP is an extension of the equal intensity model for speaker pairs. It assumes that all speakers within the system are active at all times.

Gains within the system are assumed to be normalized with constant energy (Kostadinov et al, 2010). This is shown in equation 1.1 where g is the gain at speaker n, for N total speakers.

$$\sqrt{\sum_{n=1}^{N} g_n^2} = 1$$ **Equation 1.1**

The Cartesian coordinates of the source are (x_s, y_s); for a system with N speakers, the position of the i^{th} speaker is given as (x_i, y_i). Therefore, equation 1.2 shows the distance, d_i, from source to speaker for a total of N speakers (Lossius & Hogue, 2009).

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2}$$ **Equation 1.2**

for

$$1 \leq i \leq N$$

Each speaker's gain is assumed to be inversely proportional to distance between speaker and source and all speakers are active at all times. Equation 1.3 (ibid) shows the value of the gain at speaker n; where d_n is the distance between source and speaker 'n'.

$$g_n = \frac{1}{d_n \sqrt{\sum_{i=1}^{N} {}^{1}/_{d_i^2}}}$$ **Equation 1.3**

Spatial blurring can be introduced into DBAP to minimize unwanted changes in spatial spread and colouration of a virtual source especially when it is in the same position as a speaker. To adjust for this, $r_s>0$ is introduced into equation 1.2 as shown in equation 1.4.

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2 + r_s^2}$$ **Equation 1.4**

Blur can be vertical displacement between source and speaker. As 'r' increases, the source gravitates less to any one speaker (Lossius & Hogue, 2009).

1.3.5 MULTICHANNEL LIVE SOUND

Multichannel sound is used in some live situations. The broadcast of sport uses the combination of direct (the sport itself) and ambient sources (crowd noise) to create the

atmosphere of the event. In this case the image on screen tends not to correspond, in terms of localization, to the sound as this may add confusion (Holman, 2000).

Within electroacoustic music and performance, sound diffusion is used to control the relative levels of spatial deployment (Harrison, 1998). These tend to use large multichannel systems such as the BEAST (University of Birmingham, 2013) and the HISS (University of Huddersfield, n.d).

1.4 TOUCHSCREENS, SMARTPHONES AND TABLETS

A touch screen is a display area that can detect the location of single or multiple touches and movement. It allows direct interaction with elements on screen, as opposed to indirect use of a mouse (Shneiderman, 1983), and therefore has an advantage where users require quick, intuitive and precise control.

A capacitive touch screen was first described by E.A. Johnson (1965), where he suggested its use as a computer keyboard that can have variable 'choices' or soft keys. Engineers at CERN developed and manufactured a transparent, capacitive touchscreen in 1973 (CERN, 2010).

A resistive touchscreen was created by Samuel Hurst of Elographics (backed by Siemens) in 1977 and was called the Accutouch (ELO Touch Solutions, n.d). Nimish Mehta developed the first multi-touch system in 1982 (Flexible Machine Interface), which allowed more than one contact point simultaneously (Saffer, 2009). This also combined finger pressure and graphical manipulation. The Hewlett-Packard 150, released in 1983, was the first personal computer that incorporated touch technology to position a cursor on a screen (ibid).

1.4.1 IPHONE AND IPAD

The iPhone and iPad use a mutual (projected) capacitance touchscreen to detect multitouch (Lorex, 2012). Transparent conductors are patterned into spatially separated electrodes in two layers (arranged in rows and columns). Each electrode-electrode intersection is measured in a cycle. The change in capacitance is interpolated at several intersections to attain an accurate touch point. This means mutual capacitance required a larger processing load but allows 'unlimited' multi-touch (Barett & Omote, 2010).

Capacitive touchscreens allow much 'lighter' touches as opposed to resistive touchscreens which rely on finger/stylus pressure to make a connection between conductive layers (Saffer, 2009).

The study application was designed and tested using an iPhone 4 and an iPad (Original - 1st Generation) using iOS 5.0.1. Both of these devices have a 1GHz Apple A4 processor. The iPhone 4 has random access memory (RAM) of 512 DRAM and the iPad has 256 DDR RAM. The iPhone has 802.11 b/g/n Wi-Fi and the iPad has 802.11 a/b/g/n (Apple 2012 & 2013).

1.4.1.1 APPLICATION CONTROLLERS

All the relevant, wireless iOS controllers require some form of receiver or software to interpret messages sent from the controller. The simplest controllers are similar to basic remote controls and are usually bundled with home theatre packages. Table 1.1 shows applications which represent the static button controllers available on the market.

	Category	External software or accessory	Control Protocol	Types of Control	Connection	Cost
iControlAV (Pioneer Corporation, 2013)	Basic AV system controller	• Pioneer AV • Receivers and Blu-ray Players		• Button • Surround Modes • Sweet spot • Volume level of center speaker and sub • X/Y Grid	Wi-Fi	Free
Re Universal Remote Control (NewKinetix, 2011)	Expanded Remote Control	• Dongle	Infrared (IR)	• Buttons	Dongle	Free

Table 1.1 Static button controllers.

Table 1.2 shows applications that represent the more advanced and customizable part of the controller market.

	Category	External software or accessory	Control Protocol	Types of Control	Connection	Cost
AC-7 Core (Saitara Software, n.d)	DAW/Mixing Desk style Controller	• DAWs that support external MIDI controllers	MIDI	• Fader • Pad • Button • Volume knob	Wi-Fi	£5.49
MxNM (Jameson Proctor, 2012)	Simple DAW controller	• Free Wi-Fi server. • DAWs that support midi controllers	MIDI	• Pad • Button • Slider	Wi-Fi	£2.99
TouchOSC (hexler, 2013)	Modular, user built controller	• Requires a free editor to build the controller. Requires a free MIDI 'bridge'. • DAWs that support midi controllers	MIDI, OSC	• Slider • Knob • LED • Labels • Buttons (push/toggle) • X/Y Pad	Wi-Fi	£2.99
Lemur (Liine, n.d)	Modular, user built controller	• Requires a free editor to build the controller. • DAWs that support midi controllers	MIDI, OSC	• Fader • Multislider • Multiball • Knob • Ring Area • Switches • Pads • Custom Button • Container • Breakpoint • Leds • Lemur Menu • Monitor • Signal Scope • Surface LCD • Text • 'Physics' • Scripting	Wi-Fi	£34.99
TouchAble (AppBC, n.d)	Ableton Live Controller	• Specifically for Ableton Live. • Requires a server.	MIDI	• Clip Grid • Mixer • Devices • Keys • Pads • XYZ-Pad	Wi-Fi	£17.49
MIDI Touch (Domestic Cat, 2010)	User built controller	• Requires a free editor to build the controller. • DAWs that support midi controllers	MIDI	• Faders • Rotary Knobs • Pads • XYZ Pad	Wi-Fi (or camera connection kit)	£13.99
Fantastick (Pink Twins, n.d.)	User built custom interfaces	• Max/MSP	Custom	• Very large variety of objects can be created	Wi-Fi	Free

Table 1.2 Customizable controllers

Generally, the advanced controllers' output can be mapped to control instantaneous, multi-touch and surround sound parameters in a custom user interface. The graphical user interface is kept simple and uses clear colours and shapes. However, most of the controls are mixing desk fader/knob standards. They are designed for advanced users who have previous experience with this type of technology and a clear idea of the required sonic output. These systems require partner systems, such as a DAW, to map the OSC or MIDI to appropriate controls.

1.5 Apollo Creative Ensemble

The Ensemble System was primarily designed for teachers and special needs specialists. It allows them to create interactive multisensory environments (figure 1.7) that can be customized for individuals or groups with a range of disabilities. Ensemble can also be used in other environments including exhibitions, installations and play areas (Jidkov et al, 2012).

The system is made up of sensors/switches, processing/routing and output.

Figure 1.7 A Multisensory Room

1.5.1 Ensemble Hub

The Ensemble hub (figure 1.8) is the main interface between inputs and the PC. It has four inputs for simple on/off switches (3.5mm mono jack) and 2 inputs for variable, discrete value switches (6.5mm stereo jack). It contains a 433 MHz transmitter for controlling proprietary sensory equipment and a 2.4 GHz module for wireless sensors (ibid).

Figure 1.8 The Ensemble Hub

1.5.2 INPUT SENSORS AND SWITCHES

A wide range of standard 'assistive technology' switches, tailored for a variety of needs, exist in the special needs market. These switches are operated wirelessly or by connecting directly to the hub. An important concept a music therapist attempts to develop is 'cause and effect' (see section 1.2.3) or input (movement and touch) to output (sonic, light) (Corke, 2002). As discussed in section 1.2.5, the lack of haptic feedback in non-contact sensors may prove to be problematic in this area. Wireless controls are important for ease of setup, safety of use and versatility. Often setups must be built and dismantled every session. If several players are present and have auxiliary items (such as wheelchairs), trailing cables provide a hazardous environment as well as being visually unappealing (especially in a performance situation). Sensors must be robust to handle but also provide a wide range of sensitivity to accommodate players with limited mobility (Jidkov et al, 2012).

Current sensors include:

Dice (figure 1.9) – Each side of the sensor sends data depending on orientation.

Figure 1.9 The Dice Sensor in use

Press (figure 1.10) – A sensitive pressure pad that produces a variable, pressure-dependent signal.

Figure 1.10 The Press Sensor

Tilt – Sends pitch and roll data when rotated.

Squeeze – A pressure bulb that outputs variable data when squeezed.

Floorpad – A robust switch pad.

The 'Connect' can be used to connect wired sensors such as the 'Squeeze' and 'Floorpad' wirelessly.

1.5.3 DESIGNER SOFTWARE

The designer software (figure 1.11) allows users to process data received from sensors and switches, and to route them to various outputs. It is a drag and drop environment where blocks are used to represent key components of the system. These are patched together to form a 'map' of the operation of the system. The designer software is usually used before an educational or therapeutic session or performance.

Each block has associated settings and attributes, which affect their behavior. Extra options can be manipulated such as chord/note sequences, video clip playback and lighting patterns. As well as input and output blocks, there are processing blocks which allow the user to add delays, expand the signal range and alter how the switch operates. All blocks are defined by an XML file which can be altered to produce blocks specific to an educational environment or custom installations as well as expand control via peripherals and web applications.

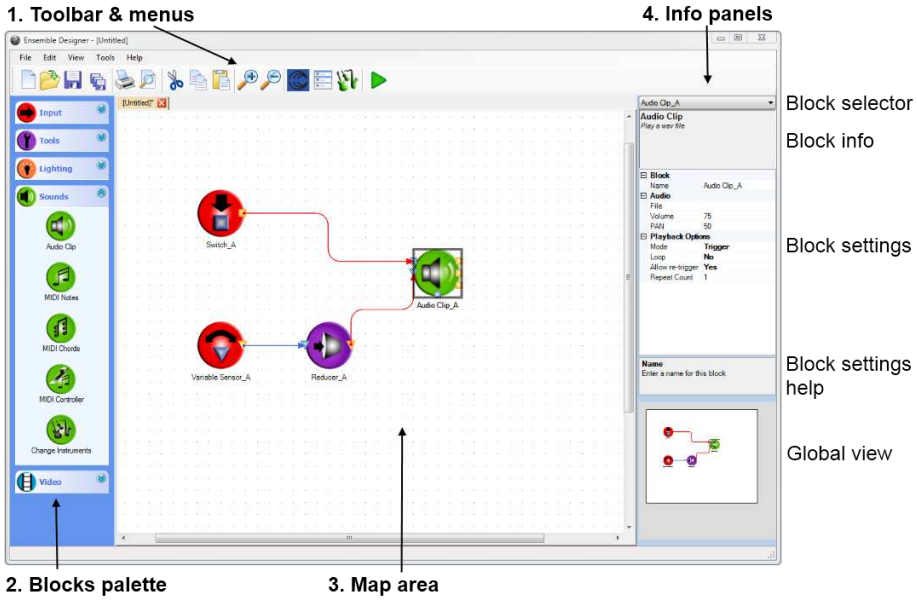


Figure 1.11 The Designer Software

1.5.4 PLAYER

The player (figure 1.12) is used during a session or performance. It is a simple Hi-Fi style interface that loads a map created in Designer. During operation, added complexity is not required and can serve as a hindrance in terms of how the session progresses and could intimidate non-technically proficient users.

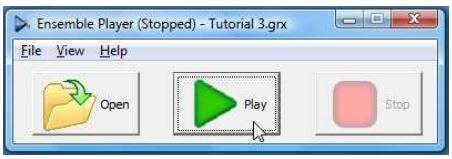


Figure 1.12 The Player

1.5.5 DEVICES MONITOR

A Devices Monitor tray application handles services. This updates automatically as devices are attached or disconnected from the system. Separate services interpret messages from the Player software and deal directly with the hardware. This means that new services can be developed for added functionality. Current services include:

DMX Output – used for lighting control; MIDI Output; Gamepad Input – any device that appears to the PC as a gamepad can be used.

SECTION 2: DEVELOPMENT

2.1 USERS AND DESIGN CONCEPTS

The aim of the study is to design and develop a prototype application to expand the control of the Apollo Creative Ensemble System. The application will focus on the surround sound implementation of the system and serve as a proof of concept for future development. It will allow users to control the movement of sound signals around an arbitrary 2-dimensional speaker setup.

The application will be tailored towards the current users of the Ensemble Hub and Apollo Creative products. These include teachers and special needs specialists such as music therapists. Additional advanced users will also be considered.

The application will send control messages in the form of volume gain instructions for the speakers. No direct signal processing will be used to synthesize sound. This restricts the type of panning that can occur, ruling out advanced spatialisation techniques such as time delays, phase and frequency correction.

2.1.1 PRIMARY USERS

Primary User Profile: teachers, special needs specialists and music therapists

Users have some familiarity with basic software but not necessarily related to music technology.

Music Technology Software	**Other Software**	**Hardware**
• Ensemble Designer	• Office Packages	• Ensemble Hub
• Media Players	• Internet Browsers	• Motion Sensors
		• Hi-Fi Systems
		• Smartphones and related devices

The iOS environment was chosen for development because the iPhone and iPad are currently very popular phone and tablet interfaces with which many users, including pupils and music therapy clients, will already be familiar or willing to familiarize themselves (Heinrich, 2012). Apple has been careful to enforce consistency throughout

applications that it distributes. This can be seen in the explicit Human Interface Guidelines (HIG) that it sets out for 'App Store' approval (Apple, 2010).

Tidwell (2006) mentions several 'patterns' that are important in interface design. These can be used as a specification outline:

Habituation: Consistency across applications and coherence with a user's 'spatial memory'.

The Ensemble Designer and Player are simplified programs for potentially non technically proficient users. Sections 1.3.3 and 1.4.1.1 show that panners often have engineer level controls and high skill users in mind. Therefore:

1. *The study application should focus predominantly on the non-technical category of users in line with Ensemble's current software.*
2. *The application's main features should be easier to use than the equivalent features in standard DAWs.*

Instant Gratification: A user must be able to accomplish an initial and/or simple task quickly and effectively. This increases confidence in the application and the user's ability.

See section 1.2.3 with regards to cause and effect.

3. *The application must have methods of operation that allow instantaneous consequences relating to the sonic output.*
4. *The application must allow a quick way of transferring sonic ideas into reality without being distracted by the underlying complications of the task. These ideas include:*
    a. *A direct movement of the sound.*
    b. *A planned 'path' that the sound will take.*

Incremental Construction: After primary 'instant gratification', the user must be able to slowly build up a more complex construction that allows midstream changes and safe exploration of the environment.

5. *The application must allow initial quick constructions to be augmented in various ways and to allow the reversal of these augmentations. Examples of this include:*
    a. *Manipulating the direction and speed of the path.*
    b. *Manipulating the location, size and orientation of the path in space.*
6. *The application should be able to operate during direct interaction but also in 'autonomous' mode, given the correct conditions.*
    a. *Direct interaction: direct movement of the sound.*
    b. *Autonomous: allow the movement of the sound to be automated.*

Gaming-style Interfaces: Noble (2012) discusses creating environments and experiences focusing on experimental models of interaction (e.g. drawing to control sound). It is worth noting that typical users do not set up their speakers according to ITUs standard and therefore speakers often have arbitrary positions dictated by the size and shape of the room.

7.
    a. *The application will use bright and simple graphics to allow quick, uncluttered interaction.*
    b. *The application will allow arbitrary positioning of speakers.*
8. *The application will have strong visual feedback in addition to sonic feedback.*

2.1.2 ADVANCED USERS

Composers and sound designers will have some of the same requirements as teachers but will require additional and higher complexity controls. Having this user group may also help to expand Apollo Creative's current market for the Ensemble system.

Advanced User Profile: composers and sound designers

Users are familiar with a range of common software and specifically that related to music technology.

Music Technology Software	**Other Software**	**Hardware**
• Ensemble Designer	• Office Packages	• Ensemble Hub
• Media Players	• Internet Browsers	• Mixing Consoles
• Digital Audio Workstations		• Outboard studio equipment (e.g. panners)
		• Motion Sensors
		• Hi-Fi Systems
		• Smartphones and related devices

9. *The application will aim to satisfy the needs of a secondary group of users: composers and sound designers.*
    a. ***Complex Sound path automation:*** *The application will provide additional layers of automation to perform complex and evolving sound trajectories.*
    b. ***Intricate Manipulation:*** *the application will allow users to manipulate paths in a complex way that can be useful for composition and performance (e.g. electroacoustic music pieces).*

2.2 System Specification

1. The application will run on iOS devices and send/receive control messages wirelessly.
2. The application will use simple colours and shapes.
3. The user will be able to change the volume of the speakers individually and in groups. Control messages will be sent as the volume is changed.

4. The user will be able to specify the position of the sound by directly touching and moving a finger (or stylus) on the touchscreen. This will send out control messages as the position is moved.
5. The user will be able to draw a sound trajectory onto the touchscreen. Once complete, it can be played back retaining the original timing. Playback will send control messages. The position of the sound will be shown on the screen.
6. The user will be able to route movement sensor inputs (sound sources) to the pan trajectories.
7. The user will be able to reverse and change the speed of the sound trajectory as it is playing.
8. The sound trajectory can be played back in a continuous loop, during which no user interaction is required, or once.
9. The user will be able to use 1, 2 and 3 finger 'touch gestures' to respectively move, resize and rotate the sound trajectory as it is playing.
10. The user will be able to draw multiple sound trajectory layers to move the trajectory below automatically.
11. The user will be able to position objects, representing speakers, arbitrarily on screen to represent the real-world speaker positions.
12. The user will be able to save and recall application projects.

2.3 System Overview

2.3.1 Development Environment

The application was designed within the XCode integrated development environment (IDE) primarily using Openframeworks (OF). Openframeworks is a C++, opensource toolkit designed for 'creative coding'. It consists of a series of software libraries and integrates with these in a software framework. The framework acts as an interface for accessing the functionality of the libraries, allowing the code to respond to events controlled by the framework (inversion of control). The programmer operates the higher-level details of the application while the framework provides low-level functionality (Openframeworks, 2012). The toolkit contains a range of add-ons that allow quick integration of iPhone application development, graphic processing and open sound control (OSC) message sending.

Figure 2.1 shows the Openframeworks code structure. Note that the filenames differ slightly for use with iOS (see section 2.3.3).

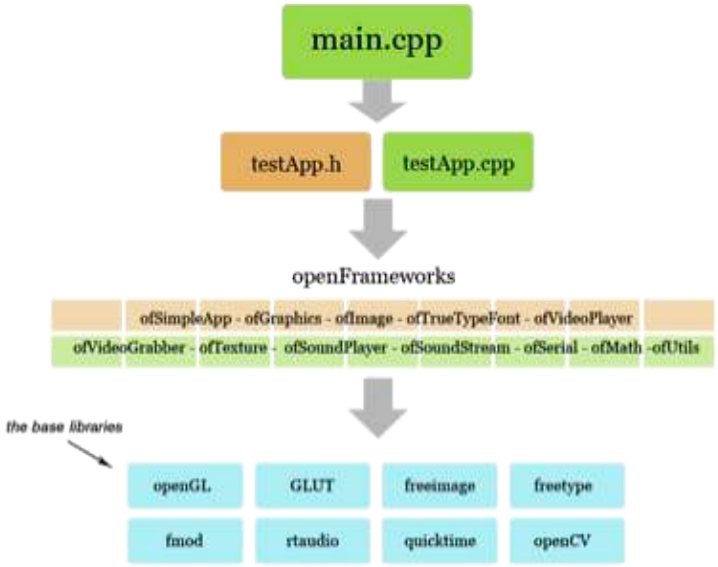


Figure 2.1 The OF Code Structure

2.3.2 PROVISIONING

Being a member of the Apple iOS Developer Program allows development of iOS applications, registration of devices and submission to the 'App Store'. A provisioning profile acts like a digital signature for applications and allows them to be uploaded and tested on devices and distributed in the App Store.

2.3.3 IOS INTEGRATION

The 'ofxiPhone' addon is an external allowing the use of Openframeworks for development on an iPhone and iPad. Figure 2.2 shows the operation of OF and 'ofxiPhone'.

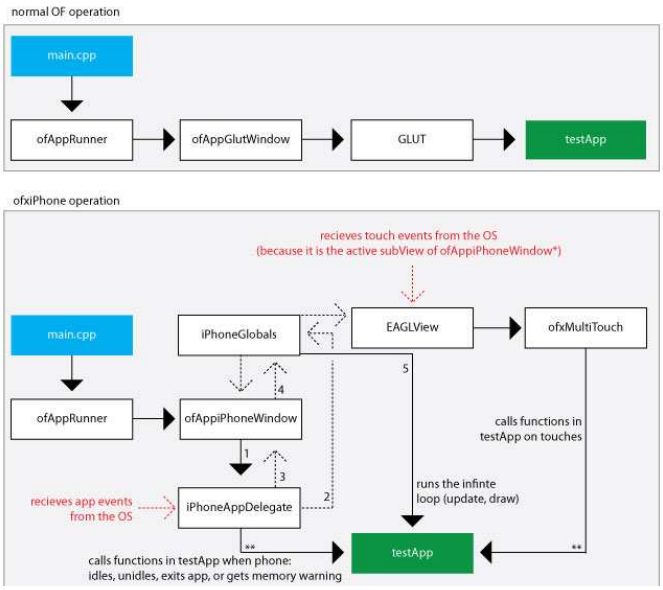


Figure 2.2 'ofxiPhone' operation

As shown, 'main.cpp' is the application delegate. It receives application events from the operating system and handles launches/quits, idles and memory warnings.

When the application starts, 'ofAppRunner' calls the 'setupOpenGL' event from 'ofAppiPhoneWindow'. 'iPhoneAppDelegate' is the main delegate for the application. The application is launched and a window with an EAGLView 'subView' is set up in iPhone Globals. An Objective-C timer is also set up which infinitely loops the 'ofAppiPhoneWindow' at a default 60 frames per second (fps). This operates the functions within 'testapp' or, in the case of the study application, 'apolloapp'. EAGLView is the main subview that receives touch events and subsequently calls appropriate functions in the external addon 'ofxMultitouch'. These touch events are passed to 'testApp' or 'apolloapp'.

The study application uses user interface (UI) elements to build menus and buttons. This requires UI calls within 'apolloapp'. For this reason the file extension for this is '.mm' rather than '.cpp'. This means that, whilst most of the coding is carried out in C++, objective-C classes can also be called. Applications built in OF in C++ can be ported to desktop operating systems easily, allowing for expansion to multi-device applications for future development.

The main C++ source file classes used for development within OF are written using two files: 'apolloapp.h' and 'apolloapp.mm'. The header file, 'apolloapp.h', contains the variables and function prototypes for the class as well as any included files and

preprocessor statements. The 'apolloapp.mm' source file is the actual class code. This file contains several applications that get called at relevant times. The relevant functions include:

setup() – This gets called once when the application is started. It is used to initialize variables, set up the GUI views and to set up other initial conditions such as multi-touch, frame rate and the OSC sender/receiver.

update() – This gets called continuously. It is used to monitor:

- The state of the application and make changes if the application is in a certain state.
- Incoming OSC messages.
- Connection status.
- Accelerometer events.

draw() – This is called continuously and is used to control drawing routines (this cannot be done in 'update()').

touchDown() – This is called when the user touches the screen of the device.

touchMoved() – This is called when the user moves their finger across the screen.

touchup() – This is called when the user releases the touch.

2.4 Application Design

2.4.1 Menu System

This section will discuss the features and components of the application.

Figure 2.3 shows how the application menus are structured. The menu system is designed to fulfill the operation specification and provide user workflow.

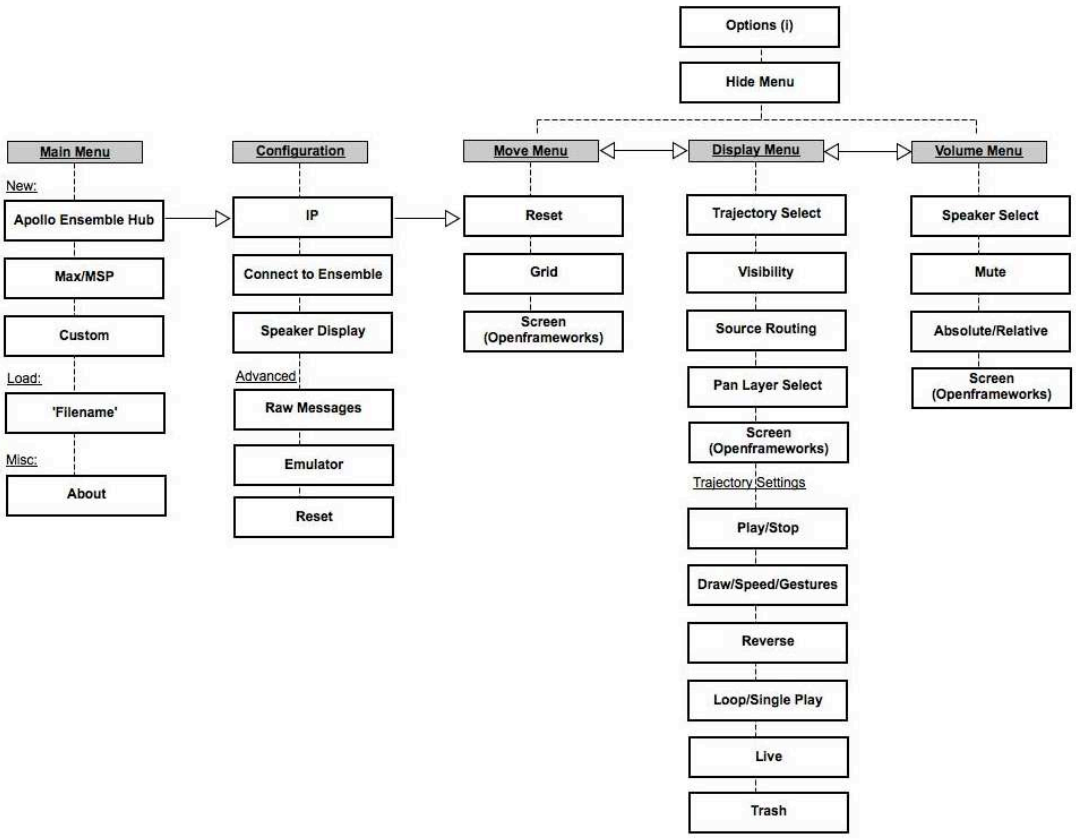


Figure 2.3 Application Menu Structure

2.4.2 FILE ARRANGEMENT IN XCODE

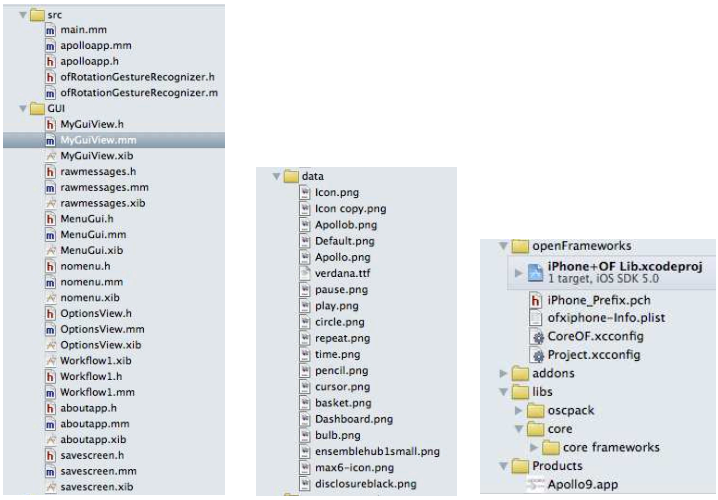


Figure 2.4 Application Folders and Files

OF folders containing source code and graphical user interface (GUI) elements, such as image files and fonts are shown in figure 2.4. They also contain OF specific files, such as: 'addons' - containing external add-ons (e.g. 'ofxiPhone'); 'libs' - containing the OF core and libraries that are used.

2.4.3 GRAPHICAL USER INTERFACE SETUP

The application toolbars and menus were designed to look similar to current applications on the 'App Store' by using Cocoa Touch objects. This required a '.xib', '.h' and '.mm' file to be created for each 'view'. Objective-C is the primary language for GUI elements. 'apolloapp.h' variables were references extensively by including the file within the '.h' file and pointing to it :

```
apolloapp *myApp.
```

Variables and functions within 'apolloapp.h' could be accessed by prepending the arrow operator 'myApp->'. Conversely, UI objects could be accessed from 'apolloapp' in a variety of ways.

The following example code shows how views were set up within 'apolloapp' in 'setup()'.

```
exampleview * exampleviewcontroller;

setup(){ //Run on startup
exampleviewcontroller=[[exampleview alloc] initWithNibName:@" exampleview"
bundle:nil];//Initialises the view
[ofxiPhoneGetGLView() addSubview: exampleviewcontroller.view];
exampleviewcontroller.view.hidden=YES;//Hides the view initially
```

This sets up an initially hidden view on load. Note that 'ofxiPhoneGetGLView()' allows touches to 'pass through' the view to the OF window.

2.4.4 GUI OBJECT LINKING

UI objects within the view are positioned within the 'xib' file. Their send event is chosen (e.g. 'Touch Up Inside' – button; 'Did End on Exit' – text field) and linked to a function in the '.mm' and '.h' file ('File's Owner'). If the UI object has a value associated with it (text field/slider), then an additional variable is linked to the object (Referencing Outlet).

2.4.5 STARTUP

When the application is started, the 'main.mm' file runs.

1. The OpenGL context is set up using the function 'ofSetupOpenGL()'. This function takes 3 arguments: screen width, height and mode. The values are

1024, 768 and 'OF_FULLSCREEN' (making the window full screen) respectively.

2. The function 'ofRunApp()' runs. This sets up the base application, timing and runs the application.

Within 'apolloapp.mm', function 'setup()' runs. The following actions are carried out:

1. The frame rate is set to 240 fps using the function 'ofSetFrameRate()'.
2. Multi-touch is set up using add-on function 'ofxRegisterMultitouch()' with argument 'this' (specifying the current application). It requires the add-on file 'ofxiPhoneExtras.h' to be included in 'apolloapp.mm'.
3. The first screen requires the filenames of saved files to be loaded. The function 'AJloadfilenames()' is run. See section 2.4.18.2.
4. The GUI views are set up (section 2.4.3). This allows them to be controlled and displayed. All views are initially hidden except 'Workflow1' - the 'New/Load' screen.
5. The background is set to black using function 'ofBackground()' with arguments '0, 0, 0' representing the RGB (red, green, blue) values.
6. OSC sender and receiver are set up. See section 2.4.7.
7. Relevant variables are initialized.
8. The device is set to be forever 'awake' by disabling the idle timer:

```
[UIApplication sharedApplication].idleTimerDisabled = YES;
```

If the application was to sleep, it would disrupt message sending. The application can still be locked and forced to sleep using the 'lock' button on the device.

2.4.6 THE NEW/LOAD SCREEN

This screen is displayed on startup (figure 2.5). The associated group of files are titled 'Workflow1'.

Figure 2.5 The New/Load Screen

1. **New**. This gives the user 3 buttons that allow them to set up a new project.
2. **Load**. This gives the user a choice of 4 buttons which can load a project if available. Each button corresponds to one slot. Initially the slots are populated using the function 'AJloadfilenames()' (see section 2.4.18.2).
3. An 'About' button is also available. This gives the user some basic information about the application.

2.4.6.1 NEW

Tapping on the 'Apollo Ensemble Hub' button takes the user to the configuration screen for the project. Within 'Workflow1.mm' the button operates IBAction 'ensemblenew' which sets global variable 'ensemblenewinstance' to true and hides the view.

Within 'apolloapp.mm' in 'update()' an if-statement monitors this variable. When true, the Configuration screen ('MyGuiView') is made visible using the dot operator 'view.hidden' and 'ensemblenewinstance' is set to false.

2.4.6.2 LOAD

The filenames are loaded using 'AJloadfilenames ()' (see section 2.4.18.2). This function populates the variable string array 'filenames[]'. To display the names on screen, within 'Workflow1.mm', function 'viewwillappear' is operated when the view is setup. This creates a temporary NSString using 'filenames[]' and then sets the title of the 4 load button labels to variable 'load0/1/2/3name'. E.g. the first button:

```
NSString *objcString0 = [NSString stringWithCString:myApp->filenames[0].c_str()
encoding:[NSString defaultCStringEncoding]];

[load0name setTitle:objcString0 forState:UIControlStateNormal];
```

Clicking on one of the loading buttons operates IBAction 'load0/1/2/3'.

2.4.6.3 ABOUT

Clicking on 'About' operates IBAction 'about' which sets global variable 'aboutinstance' to true. In 'apolloapp.mm' in 'update()' this makes the 'aboutapp' view visible.

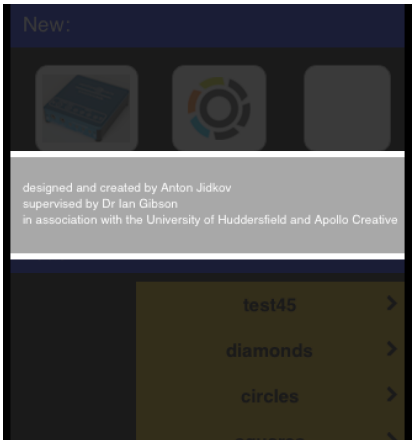


Figure 2.6 The About Screen

This view displays a dialog box with some basic information about the application as shown in figure 2.6. The 'aboutapp' view background is made translucent by setting the 'alpha' level to 55%. A transparent button covers the entire screen; when the screen is tapped the dot operator 'view.hidden' on the view is set to 'true', hiding the view.

2.4.7 OPEN SOUND CONTROL SETUP

Control messages are sent out of the system in the form of OSC messages. This is implemented using the 'ofxOsc' add-on created by hideyukisaito (GitHub, 2013). The OSC receiver and sender are initialized in 'apolloapp.mm' in function '*setup()*':

```
        receiver.setup(PORT);
        sender.setup(ipip, PORT);
```

A default port, 12345, is set up. The IP address is set via the user input in the configuration menu. The OSC receiver in 'update()' in 'apolloapp.mm' constantly scans for messages sent to the device IP:

```
while( receiver.hasWaitingMessages() ){ //receiver while loop

            ofxOscMessage m; //temporary message variable created
            receiver.getNextMessage( &m ); //next message is requested

                msg_string = m.getAddress(); //address is requested

            …
             //Formatting of messages occurs depending on type
            …

             // adds the message to the string array
             msg_strings[current_msg_string] = msg_string;
             current_msg_string = ( current_msg_string + 1 ) % NUM_MSG_STRINGS;
 }
```

Received messages can be viewed from the 'Configuration' menu if the 'Raw Messages' button is tapped. These are displayed by hiding the configuration view using 'myGuiViewController'. This allows the user to view all messages coming into the system, which can be useful for debugging and status checks:

```
if(menuset==0){ // OSC MESSAGE DISPLAY

        ofSetColor(255, 255, 255);// white lines are drawn
        ofLine(0, 112, 300, 112);
        ofLine(0, 340, 300, 340);

        ofSetColor(225, 58, 135);

       // received messages are displayed
        for(int i=0; i<NUM_MSG_STRINGS; i++)
        {
            verdana14.drawString(msg_strings[i], 10, 130+i*20);
        }
    }
```

Figure 2.7 shows the received messages displayed. In this example, there are 5 speakers and 5 sources.

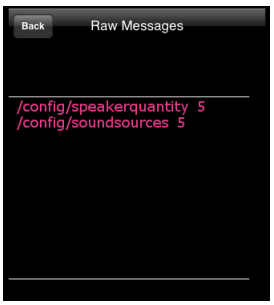


Figure 2.7 Raw Messages

2.4.7.1 THE CONFIGURATION MENU

In Configuration (figure 2.8), users are guided through the process of entering the IP address of the PC, establishing a connection and finally displaying the speakers.

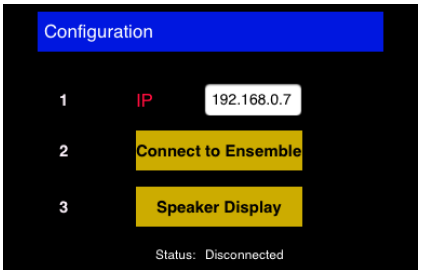


Figure 2.8 Configuration

The configuration menu is within the 'MyGuiView' files in the GUI folder. When the user enters a value in the text box:

1. The IP address is stored in variable 'ipip' (in the form of a UTF8String rather than an NSString).
2. The OSC sender 'guisender' is set up with that IP address and the default port 12345.
3. The keyboard (FirstResponder) is resigned (hidden). The keyboard is also hidden with a 'background tap'. This is done by setting up a 'Tap gesture recognizer' in 'viewDidLoad' in 'MyGuiView.mm':

```
-(void)viewDidLoad {

    UITapGestureRecognizer *viewBackgroundTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(backgroundTap:)];
    viewBackgroundTap.cancelsTouchesInView = NO;
    [self.view addGestureRecognizer:viewBackgroundTap];
    [viewBackgroundTap release];
}
```

When the user taps 'Connect to Ensemble', the following actions are carried out:

```
-(IBAction)button: (id)sender{
//OF menu mode is set to Raw Messages
    myApp->menuset = 0;
//Temporary OSC message created
    ofxOscMessage buttonmessage;
//Address is set
    buttonmessage.setAddress("/getconfig");
//Message is sent
    guisender.sendMessage(buttonmessage);
//The current set of received messages is cleared
    myApp->current_msg_string=0;
    for( int i=0; i<NUM_MSG_STRINGS; i++ ){

                myApp->msg_strings[i]="";
        }
    myApp->speakerdisplayload=0;
    myApp->emspeakerdisplayload=0;
}
```

Received messages are in the form - speaker quantity: '/config/speakerquantity s' - where s = speaker quantity; user inputs: '/config/soundsources i1' - where i1= the amount of input sources (e.g. motion sensors). If the correct messages are received the status is set to 'Connected' in the 'apolloapp.mm' file. Note that the controller variable is accessed using the dot operator:

```
myGuiViewController.constatus.text=@"Connected";
```

Finally the user taps 'Speaker Display' to view the system setup. This sets 'menuset' to 1 and hides the configuration view. If this is the initial load (i.e. the user has started a new project), the speaker and input quantities are read from 'msg_string[]' and the system is populated with speakers in a grid (section 2.4.8).

'Advanced' options give the user the ability to set speakers and sources without exchanging configuration messages. This is useful if there are any initial communication problem and for demonstration and debugging.

As mentioned, the application changes between the active OF windows monitoring the status of the 'menuset' variable where values are as follows:

'menuset' Value	**Openframeworks Window**
0	Raw Messages
1	Move Menu
2	Display Menu
3	Volume Menu

2.4.8 THE MOVE MENU

This view is shown after the Speaker Display button is tapped in the configuration menu. Initially, the user is presented with a grid of speakers, corresponding to the configuration messages, and a listener position, which can be disabled in the options menu. Figure 2.9 shows a 5-speaker setup.

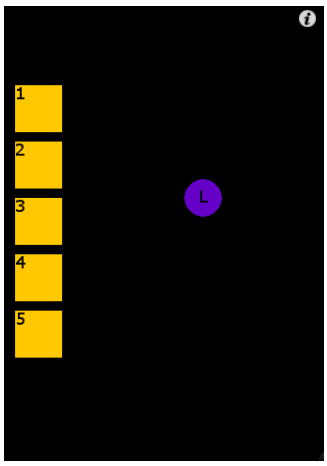


Figure 2.9 Move Menu (no toolbars)

Whilst in the Move, Display and Volume menus, the user is given the option to display/hide the toolbars by tapping the 'i' button on the top right to display, and 'X', when visible, to hide. This feature is implemented using set of GUI files 'nomenu'. The 'X' is a 'bar button item' with the identifier 'stop' located in the 'MenuGui' GUI files:

```
//.h
-(IBAction)hidebutton: (id)sender; //Hide button initialized

//.mm
-(IBAction)hidebutton: (id)sender{ //Hide button operated
    myApp->menuvis=false;
    myApp->menuvisfalseinstance=true;
}
```

Note that variables 'menuvis' and 'menuvisfalseinstance' in 'apolloapp' are being altered. Hiding and showing views is implemented within 'apolloapp.mm':

```
        //Draw the toolbars
        if(menuvis==true && optionsmenu==false){
            MenuGuiController.view.hidden = NO;
            MenuGuiController.speakerlabel.text=speakersNS;
            MenuGuiController.sourcelabel.text=sourcesNS;
            nmcontroller.view.hidden= YES;
            OptionsViewController.view.hidden=YES;
        }
        //Hide the toolbars
        if (menuvis==false && optionsmenu==false) {
            MenuGuiController.view.hidden = YES;
            nmcontroller.view.hidden= NO;
            OptionsViewController.view.hidden=YES;
       }
```

When the menu is being hidden, 'menuvis' is set as false and therefore the menu ('MenuGuiController') and the options menu (as a safety precaution) are hidden and the 'nomenu' view ('nmcontroller') is displayed.

The 'nomenu' fileset simply displays one button of type 'info light' that reverses the hiding processes by setting 'menuvis' to true. This provides an unobtrusive interface for navigating a toolbar-less screen. This is good for showing a clear interface and appropriate for live performance where additional screen elements are not required. It is also a safety feature preventing accidental taps that may drastically change the functionality of the application.

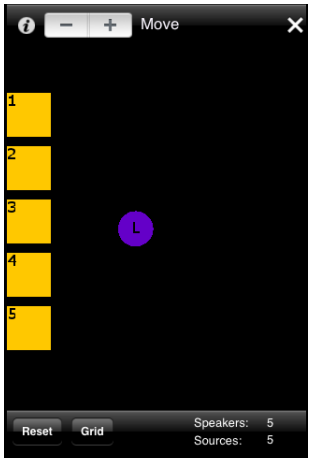


Figure 2.10 Move Menu (with toolbars)

When the toolbars are displayed (figure 2.10), the top toolbar provides information about the whereabouts of the user; a stepper, to move between menus; an options button ('i' on the top left) and the 'X' button to hide the menu. The bottom toolbar gives the user functionality options and displays status information about speaker and source quantity.

The initial grid layout is determined within the 'myGuiView' fileset in the GUI folder:

1. A for-loop is set up that sets the coordinates of the speaker display rectangles to the array 'rectcoordinates[]'.
2. Every 5 speakers, a new row is started. A maximum of 25 speakers is allowed.
3. These are drawn in 'apolloapp.mm' in 'draw()'.
    a. Here, a for-loop is set up to run through the sound outputs.
    b. The colour is set to yellow ('ofSetColor(255,200,0)' using RGB values).
    c. The function 'ofRect()' is called with the 'rectcoordinates[]' used as the coordinates to draw the rectangles.
    d. A speaker number is drawn within the rectangle to identify it. This is done by:
        i. Setting the colour to black.
        ii. Converting the sound output integer to a string using 'ofString()'.
        iii. Drawing the string using 'verdana14.drawString()' using offset 'rectcoordinates[]' for the position to display the number inside the speaker rectangle.

Verdana14 is an 'ofTrueTypeFont' which is initialized in 'setup()' with the following values:

```
verdana14.loadFont("verdana.ttf", 18, true, true); //font loaded from the data file
    verdana14.setLineHeight(28.0f); //line height set
    verdana14.setLetterSpacing(1.0); //letter spacing set
```

This allows a font to be bundled with the application, shown in the data folder as a '.ttf' file.

2.4.8.1 RESET BUTTON

The 'Reset' button on the bottom toolbar carries out the speaker layout operation again. This means that, if the speaker display icons have been moved, they are reset to initial conditions.

2.4.8.2 GRID BUTTON

The 'Grid' button displays a grid in the OF window that helps the user align speakers and the listener position. The button operates Boolean variable 'grid' monitored in 'apolloapp.mm' in 'draw(). This variable operates the function 'AJgriddisplay()' located in 'apolloapp.h':

```
void AJgriddisplay(int xoffset, int yoffset){
        ofSetColor(160, 160, 160); // set colour to grey
        // draw vertical lines across the screen
        for (int i=xoffset; i<xoffset+340; i+=10) {
            ofLine(i, yoffset, i, yoffset+485);
        }
        // draw horizontal lines across the screen
        for (int i=yoffset; i<yoffset+490; i+=10) {
            ofLine(xoffset, i, xoffset+330, i);
        }
}
```

This function operates by calling the function 'ofSetColor()' to set the colour to grey; it then runs two for-loops to draw the vertical and horizontal lines. It uses the negative of the 'ofmovemodetranslate[]' array (arguments 'xoffset' and 'yoffset'), which stores the x/y coordinates of the OF screen translation as the offsets (see section 2.4.8.6) to determine where to start the grid. The function 'ofLine()' is called to position the lines regularly as shown in figure 2.11 creating a perpetual grid. The grid is always visible no matter how far the user translates the OF window.

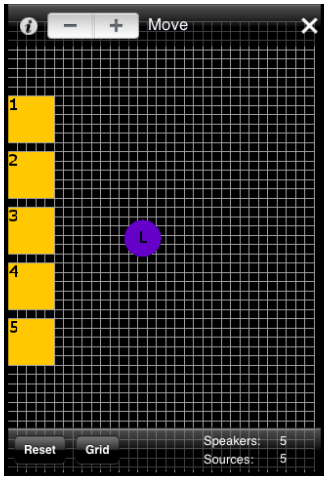


Figure 2.11 Move Menu (with gridlines)

The grid lines force the speakers and listener to snap to the grid as well as providing a visual reference. This moves the speakers, listener position or translates the OF window as normal but sends the final coordinates through function 'Ajgridsnap()':

```
int Ajgridsnap(bool xory,int x, int y){
        //xory: true=x, false=y
        //makes x and y multiples of 10
        x=x-(x%10);
        y=y-(y%10);

        //returns the updated coordinates
        if (xory==true) {
            return x;
        }
        else{
            return y;
        }
```

For the x and y coordinates separately, the function determines how far away the coordinate is from a multiple of 10 and subtracts this number, forcing the final coordinate to be a multiple of 10, thus 'snapping' to the grid.

The quantity of speakers and input sources is also displayed on the bottom toolbar. This is done by taking the 'NSString' variables 'speakersNS' and 'sourcesNS' (set in 'myGuiView.mm' by 'guispeakers.text') and setting variables within 'menuGui.mm':

```
speakerlabel.text=myApp->speakersNS;
sourcelabel.text=myApp->sourcesNS;
```

2.4.8.3 ID PAIRING

The main function of the Move menu is to position speakers and the listener on screen. Coordinates of touches and movement arrive in 'apolloapp.mm' into 'touchdown()', 'touchmoved()' and 'touchup()'. They are orientated as shown in figure 2.12.

Figure 2.12 iPhone coordinate orientation

When a user touches down on a speaker or listener, the id (order) of the touch is paired with the object. The first finger is id=0, the second id=1 and so on. Ids are reset when the finger is lifted. With 'menuset'=1 (move menu) in 'touchdown()', if both x and y coordinates of the touch are found to be within the boundaries of the speaker, the id is recorded in array 'speaker_id[]'. This is carried out in a similar way for the listener position saving the variable 'listener_id':

```
for( int i=1; i<=soundoutput; i++ ){ //Runs through all the speakers
            if(rectcoordinates[(i*2)-2]<=x && x<=rectcoordinates[(i*2)-2]+50){
                if(rectcoordinates[(i*2)-1]<=y && y<=rectcoordinates[(i*2)-1]+50){
                    speaker_id[i-1]=id; //record the id if the coordinate is within
                                                the speaker
                }
            }
        }
        //Listener ID paring
        if(x>=listenercoordinates[0]-10 && x<=listenercoordinates[0]+10 &&
y>=listenercoordinates[1]-10 && y<=listenercoordinates[1]+10){
            listener_id=id;
        `
```

In 'touchup()', the speakers and listener are unpaired by setting the id number in the array to '-1':

```
for( int i=1; i<=soundoutput; i++ ){ //runs through all the speakers
            if (speaker_id[i-1]==id) {
                speaker_id[i-1]=-1;//unpairs the relevant id
            }
        }
        //Listener/ID de-pairing
        if (listener_id==id) {
            listener_id=-1; //unpairs the id
        }
```

Once pairing has occurred, the user will move their finger to move the object. In 'touchmoved()':

1. A for-loop runs through the objects.
    a. If the id of the moved finger matches a speaker (or listener) id:
        i. An object movement is registered.
        ii. The coordinates are updated.

Pairing is carried out to avoid 'dropped' objects when the user moves their finger quickly. At fast speeds, coordinate messages returned to OF become sparser and therefore the user's finger coordinates may end up outside of the object before coordinates are updated, causing it to be dropped.

2.4.8.4 MOVING

After id pairing, the user moves their finger to position the object. This updates array 'rectcoordinates[]'. The rectangle size is 50 pixels. Therefore move coordinates are offset by 25 pixels to center the rectangle on the finger. This is not done for the listener position because 'ofCirc()' draws from the centre, as opposed to 'ofRect()', which draws from the top left corner:

```
for( int i=1; i<=soundoutput; i++ ){//Monitors speakers
            if (speaker_id[i-1]==id) {//Movement has occurred inside a speaker
                speakersaremoving=true;

               //update rectangle coordinates
                    if (grid==1) {//if the grid is on, snap to grid
                      rectcoordinates[(i*2)-2] = Ajgridsnap(true, x-25, y-25);
                      rectcoordinates[(i*2)-1] = Ajgridsnap(false, x-25, y-25);
                    }
                    else{
                        rectcoordinates[(i*2)-2] = x-25;
                        rectcoordinates[(i*2)-1] = y-25;
                    }
```

The listener position is moved in a similar way. Movement operations can occur in multi-touch. Therefore several objects can be id paired and moved at once.

2.4.8.5 PREVENTING OVERLAP

To keep movement clear and controlled, objects cannot overlap. This is handled by the 'AJoverlap()' function located in 'apolloapp.h' and called after the update of 'listenercoordinates[]' and 'rectcoordinates[]' occurs.

This function has 2 arguments: the moving speaker integer and a bool determining what the moving object is (true=speaker, false=listener). Within 'apolloapp.mm', a for-loop is run using 'AJoverlap()' to determine any object overlap. Within 'AJoverlap()', for speaker movement:

1. A variable 'start' is set to i+1 (the next speaker).
2. A for-loop is run from the variable 'start' through the rest of the speakers.
3. Using if and else conditionals, the function compares the coordinates of the moving speaker (x/y) to the current speaker in the for-loop (x1/y1). First it compares the overall coordinates. If both the x and y coordinates are found to be within each other, the function will determine which side is overlapping: right, top, bottom or left.
4. To avoid the speaker looking for an overlap with itself, the for-loop makes sure that, when the count = i (the moving speaker), it sets the x1/y1 variables to the listener coordinates to check for an overlap. In this way all objects are checked.
5. After this final check, the for-loop is requested to break.
6. 'Rectcoordinates[]' are updated as required leaving a 1 pixel gap between the object edges.

If the listener is being moved, the steps taken are very similar except the for-loop only runs through the speakers.

2.4.8.6 WINDOW TRANSLATE

The user can explore the display area in order to position speakers further away or to allow a clearer view of the space. In the 'Move menu', when the user moves a finger outside of the speakers and listener objects, the entire OF window moves (including all objects and paths). This is carried out in the 'apolloapp.mm' file:

1. **ofPushMatrix()** – This occurs in draw() and pushes the values of the OF matrix. I.e. saves the untranslated coordinate system.
2. **ofTranslate()** – Following the push, this function runs continuously to translate the OF window by coordinates 'movemodetranslate[0]' and 'movemodetranslate[1]' or x and y respectively
3. **ofPopMatrix()** – Finally this function restores the untranslated coordinate system.

In 'touchmoved()', in the same section as object movement is handled, if neither the listener nor the speakers are being moved and the id=0 (first finger), window translate operations occur. This is done by updating the 'movemodetranslate[]' x/y coordinates:

```
if (listener_id!=id && i==soundoutput && speakersaremoving==false) {

        if (id==0) {//First finger

                if (grid==true) { //Update the coordinates, snap to grid if
                                        required
                        movemodetranslate[0]=Ajgridsnap(true,
movemodetranslateold[0]+moveid0x-touchid0x, movemodetranslateold[1]+moveid0y-
touchid0y);
                         movemodetranslate[1]=Ajgridsnap(false,
movemodetranslateold[0]+moveid0x-touchid0x, movemodetranslateold[1]+moveid0y-
touchid0y);
                    }
                else{
                        //Update the coordinates
                       movemodetranslate[0]=movemodetranslateold[0]+moveid0x-
touchid0x;
                       movemodetranslate[1]=movemodetranslateold[1]+moveid0y-
touchid0y;
               }
       }
```

If the grid is not enabled, the value of 'modemodetranslate' becomes:

[old value] + [amount moved in x/y direction] – [initial touch position]

The translate function uses a vector translate operation. Therefore, the translate operation must be done from the origin (0,0). This is why the initial touch position is subtracted from the 'movemodetranslate' value. The array 'movemodetranslateold[]', the old value, is updated when a 'touchup()' operation occurs on id=0; 'modeid0x/y' is updated at the start of 'touchmoved()' (to the x/y coordinates) and 'touchid0x/y' are updated at the start of 'touchdown()'.

If the grid is enabled, the value of 'movemodetranslate[]' is sent through function 'Ajgridsnap()' (section 2.4.8.2).

If a window translate operation has occurred, 'movemodetranslate[]' is not equal to zero, all input touches must be altered. Function 'OfTranslate()' only affects how the window is displayed without altering variables. Therefore touches do not correlate to the updated display, with coordinates being relative to the physical touchscreen. These touches are offset appropriately at the top of 'touchdown()', 'touchmoved()' and 'touchup()'.

2.4.9 THE STEPPER

Once the user has finalized the positions of the speakers and listener, these can be locked in place in order to start manipulating the volume and panning.

To change which menu is displayed, the 'stepper', on the top toolbar to the left of the menu title, is used. This allows users to 'step' forwards and backwards continuously through the menus. Note the Move, Display and Volume menus are all located within the same view. When 'increment' or 'decrement' is pressed, it carries out operations in –'(IBAction)menustepbutton' in 'MenuGui.mm'.

1. The current value of 'menustep' (using the dot operator 'value') determines which menu should be displayed (1=Move, 2=Display, 3=Volume).
2. The relevant toolbars, buttons and labels are made visible and all others, hidden.
3. Relevant variables are updated. E.g. 'menuset' is set to the correct value.
4. The title label ('menulabel.text') is changed to the appropriate menu title.
5. Menu-specific operations are carried out. E.g. for the move menu, all speakers selected in volume mode are deselected.

2.4.10 VOLUME MENU

When the stepper value is equal to 3, 'menuset' is 3, the volume menu is displayed. This makes the 'volumetoolbar' (top toolbar) and 'volumebottomtoolbar' visible. The top toolbar contains selection buttons and the mute button. The bottom toolbar contains the slider, used to adjust volume, the 'None' selection button and the absolute/relative mode button selection. The initial state of the volume slider (variable 'volslider') is disabled because no speakers are selected. At this point, speakers and listener are locked in place.

The user can tap speakers to select (causing them to turn orange) and deselect them or press buttons to select speakers depending on their position on screen. Figure 2.13 shows the volume menu with the right speakers selected.

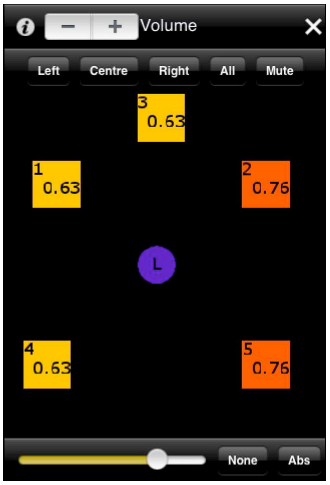


Figure 2.13 Volume mode with right speakers selected

Speaker volume is stored in the array 'speakervolume[]' (a float, 0-1). The index of the array corresponds to the speaker number. This is drawn within the speaker in 'apolloapp.mm' in function 'draw()'. A for-loop runs through the available speakers and draws the verdana14 string:

```
verdana14.drawString(ofToString(speakervolume[i-1],2), rectcoordinates[(i*2)-2]+10,
rectcoordinates[(i*2)-1]+35);
```

Note that the 'ofToString()' function is an OF function that converts integers, in this case stored in the 'speakervolume[]' array, to a string. The string is drawn in the appropriate rectangle using 'rectcoordinates[]' with an offset of x+10 and y+35.

A tap selects/deselects a speaker. Within 'touchdown()', a for-loop runs through the available speakers. Conditionals check whether a touch has occurred within a speaker. If this is the case, the speaker is either selected or deselected. The selection state of the speakers is stored in the bool array 'selectedspeakers[]':

```
for( int i=1; i<=soundoutput; i++ ){ //run through the speakers
               //determine if a touch has occurred inside a speaker
            if(rectcoordinates[(i*2)-2]<=x && x<=rectcoordinates[(i*2)-2]+50 &&
mute==false){
                if(rectcoordinates[(i*2)-1]<=y && y<=rectcoordinates[(i*2)-1]+50){
                    if(selectedspeakers[i-1]==true){
                       //if selected, deselect the speaker
                        selectedspeakers[i-1]=false;
                    }
                    else{
                       //if deselected, select the speaker
                        selectedspeakers[i-1]=true;
                        colouredit=100; //change the speaker colour
                       //make sure the volume slider jumps to the speaker volume
                        MenuGuiController.volslider.value=speakervolume[i-1];
                       //enable the volume slider
                        MenuGuiController.volslider.enabled=true;
                    }
                }
            }
```

If a speaker is selected, the volume slider ('volslider') is enabled to allow editing and setting of the volume of that speaker using the float array 'speakervolume[]'. Initially, all volumes are set to 0. A conditional within 'apolloapp.mm' function 'draw()' deals with colouring the speakers as required.

The selection buttons can give the user a quick selection of blocks of speakers depending on their position. This is useful in calibration and setup, especially in non-standard rooms.

Each of these buttons is operated in 'MenuGui.mm'. A for-loop runs through all available speakers and if the x coordinates of the speaker are less than 100, they are selected, and the rest are deselected. A similar process is in effect for the rest of the buttons. E.g. Left button:

```
-(IBAction)leftbutton:(id)sender{
    for( int i=1; i<=myApp->soundoutput; i++ ){ //run through the speakers
        myApp->selectedspeakers[i-1]=false; //deselect all the speakers
        myApp->colouredit=100; //change their colour accordingly
        //determine if speakers are on the left
        if(myApp->rectcoordinates[(i*2)-2]+25<100){
            myApp->selectedspeakers[i-1]=true; //if they are, select them
            volslider.enabled=true; //enable the slider
            volslider.value=myApp->speakervolume[i-1]; //update the slider value
        }
        else{
            myApp->selectedspeakers[i-1]=false; //otherwise, deselect the speakers
        }
    }
}
```

The Absolute/relative button affects the way that the volume is changed. Tapping the button changes between absolute and relative mode, setting the bool 'volabsolutebutton' to true and false respectively.

A for-loop runs through all available speakers and affects those that are selected. In absolute mode, when the 'volslider' value is changed, all selected speaker volumes become the value of the 'volslider' (acquired using the dot operator 'value'). As the value of 'volslider' changes, 'speakervolume' is incremented or decremented by 0.01. If relative mode is active ('volabsolutebutton'=false), selected speaker values are incremented/decremented individually. This keeps relative volumes between speakers. Constraint happens if the value of 'volslider' is at the top or bottom extreme.

As the volume is changed, OSC messages are sent with volume information. See section 2.4.19.1.

The 'Mute' button mutes all speakers. If it is selected, it disables all other buttons in the volume menu and sets its own tint to grey. It sets the variable 'mute', located in 'apolloapp.h', to true, 'muteinstance' to true and operates the function 'AJmute()'. 'AJmute()' carries out the following actions if 'muteinstance'=true:

1. Sets up a for-loop for all available speakers.
2. Stores the values of 'speakervolume[]' in 'speakervolumehold[]'.
3. Sets the value of 'speakervolume[]' to zero.
4. Deselects all speakers.
5. Sends an OSC message of the zeroed 'speakervolume[]'.
6. Sets 'muteinstance' to false.

If the mute button is deselected then 'unmuteinstance' is set to true and the above actions are carried out in a similar way except values are retrieved from 'speakervolumehold[]'.

2.4.11 DISPLAY MENU

When the stepper value is equal to 2, the display menu is displayed (figure 2.14). This is the main menu screen used for pan trajectory control. The toolbars 'displaytoolbar' (top toolbar), 'displaybottomtoolbar' and 'displaybottomtoolbar2' are shown as well as the secondary pan layer stepper on the title toolbar.

The two bottom toolbars show the 3 available pan trajectories, A, B and C, with their visibility button underneath (light bulb icon) and the sound sources on the right that can be routed to each trajectory. The top toolbar displays buttons that can be used to control the layers: play/stop, draw/speed/touch, reverse, single/loop, live mode and delete.

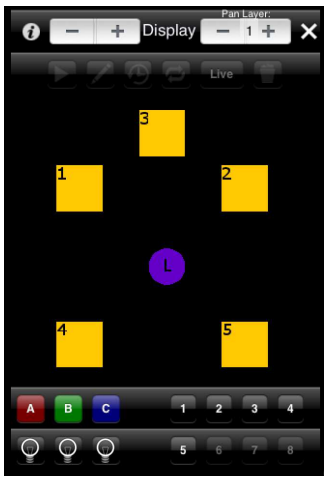


Figure 2.14 The Display Menu

The user can select a pan trajectory, highlighting the button:

1. 'selA/B/C', 'panA/B/C' and 'visA/B/C' are set to true for the relevant pan trajectory (A/B/C) and the rest set to false.
2. The button 'tint' becomes brighter, the rest darker e.g. A selected:

```
Abutton.tintColor=[UIColor colorWithRed:1.0 green:0.0 blue:0.0 alpha:0.0];
Bbutton.tintColor=[UIColor colorWithRed:0.0 green:0.6 blue:0.0 alpha:0.0];
Cbutton.tintColor=[UIColor colorWithRed:0.0 green:0.0 blue:0.6 alpha:0.0];
AVisbutton.tintColor=[UIColor colorWithRed:1.0 green:1.0 blue:0.0 alpha:0.0];
```

   Note the visibility button becomes yellow.
3. If Live mode is disabled, all the top toolbar buttons are enabled. If Live mode is enabled, only the live button become enabled and highlighted.

The sound input buttons are enabled depending on the quantity set up via the initial configuration messages. Buttons are enabled within the 'menustepbutton'. E.g. Source 1:

```
if (myApp->inputsources>0) {
            source1.enabled=true;
}
```

This continues for 8 sources, the maximum that is available in the current prototype.

If a pan layer is selected, sound sources can be routed to it allowing relevant OSC messages to be sent. The sound inputs become the colour of the pan trajectory they are routed to, providing visual feedback. E.g. 'source1' tapped with pan trajectory A selected:

```
if (myApp->selA==true) {//A is selected
        if (myApp->inputsource[0]!=1) {//if unrouted previously
            source1.tintColor=[UIColor colorWithRed:1.0 green:0.0 blue:0.0
alpha:0.0]; //set the colour to red
            myApp->inputsource[0]=1; //route the source
        }
        else{
            source1.tintColor=[UIColor colorWithRed:0.0 green:0.0 blue:0.0
alpha:0.0]; //otherwise, unroute the source and paint it black
            myApp->inputsource[0]=0;
            myApp->AJzerogain(0); //send out a zero gain message for the source
        }
    }
```

Note that the array 'inputsource[]' is used to record routing. The index is related to the source and the integer value is the pan trajectory (0=not routed, 1=A, 2=B, 3=C). Tapping the source again or tapping it with no pan layer selected de-routes it. This sets the 'inputsource[]' value to zero and runs the function 'AJzergain()' setting all gain values of that source to zero (section 2.4.19.1).

2.4.12 Live Mode

The simplest method of positioning sound is Live Mode. This option is useful for direct sound movement and speaker calibration. Figure 2.15 shows live mode with source 3 routed to pan trajectory C moving between speakers 2 and 3. If a path has been drawn, it is saved and hidden whilst view mode is active. A short tail is displayed showing the previous pan positions.

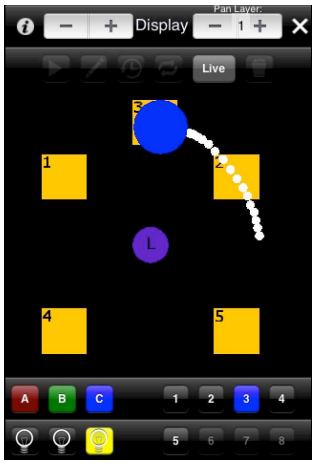


Figure 2.15 Live Mode

Clicking the live button, highlights the button and disables the other top toolbar buttons as they are now irrelevant. It also sets the variable 'A/B/Clive' to true.

When the user touches and slides in the OF window: in 'apolloapp.mm' 'touchmoved()', if id=0 (only first finger touches are allowed), 'selA/B/C' and 'A/B/Clive' are true. The following actions are taken:

1. The array 'pancoordinates[]', which holds the coordinates for live pan mode, is updated with the x/y coordinates at index 'livepan' (starting at 0 and reset at 50 – limiting the 'tail').
2. The variable 'livepan' is incremented by 2 (x and y coordinates).
3. The contents of array 'A/B/Cstopcirclive[]' are updated to the previous value of 'touchmoved()' x1/y1.
4. Gain calculations are carried out (section 2.4.19.2):
    a. The absolute distance between each speaker and the sound position are stored in an array using function 'AJabsdiff()'.
    b. Gains are calculated using array 'AJDBAPgain()'.
    c. Gains are sent via OSC for each speaker using 'AJdbaposcgain()'.

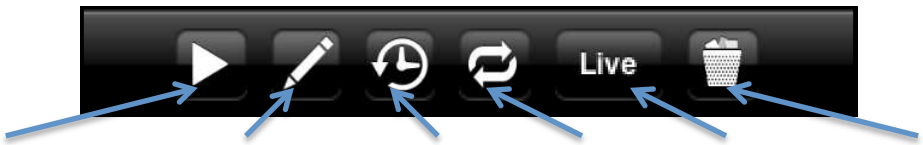
The array 'pancoordinates[]' is used to draw the current position circle and the 'tail' that appears as the path is drawn across the screen. The colour of the tail is set to white and a circle is drawn using 'ofCirc()' using 'pancoordinates[]' with a radius of 5. A circle is drawn using coordinates 'A/B/Cstopcirclive' with a radius of 30. Pan layer A is shown:

```
if(menuset==2 && (Alive==true || Blive==true || Clive==true)){
            for(int i=0; i<1000; i=i+2){
               ofSetColor(255,255,255); //set the colour to white
               ofCircle(pancoordinates[i], pancoordinates[i+1], 5); //draw the
                                                                     tail circle

                if(Alive==true && visA==true){
                   ofSetColor(255,0,0);//set colour to red
                  ofCircle(Astopcirclive[0], Astopcirclive[1], 30); //draw the
                                                                   sound circle

               }
           }
```

Note that 'pancoordinates[]' has 1000 elements but 'livepan' is limited to 50 allowing 25 'tail' points. Therefore, the length of the tail can be extended.

2.4.13 DRAW MODE

Panning can be automated by drawing a path. These can be manipulated when they are playing or paused. The speed and direction of the path can be changed; looping can be turned on and off and touch gestures can be used to further manipulate the path. Figure 2.16 shows the functionality of the toolbar buttons.



Play/Stop Draw/Speed/Touch Reverse Loop Live Mode Clear

Figure 2.16 Draw Mode Buttons

To draw a path, the 'draw/speed/touch' button should be set to draw mode (pencil icon). Clicking this button changes the mode and icon displayed in the button continuously. Three modes are available: draw, speed and touch as shown in figure 2.17.

Draw Speed Touch

Figure 2.17 Draw/Speed/Touch Button

Tapping this button operates 'drawbutton' in 'MenuGui.mm'. This cycles through values of variable 'drawbuttonsetting' (0=Draw, 1=Speed, 2=Touch). It also updates the icon as shown figure 2.17 giving the user feedback as to which mode they are in:

```
drawb.image=[UIImage imageNamed:@"Icon.png"];
```

Icons are stored in the 'data' file as '.png' files.

2.4.13.1 DRAWING

Within draw mode, to draw a path, a pan trajectory must be selected. Then, the user can draw the path directly in the OF window. Figure 2.18 shows an elliptical shape drawn in pan trajectory A. The colour of the path corresponds to the colour of the layer for identification.

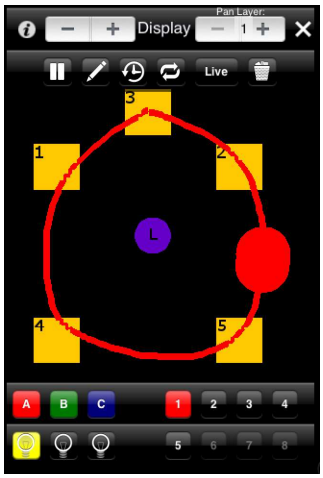


Figure 2.18 Trajectory A

The user touches the screen. In 'apolloapp.mm' in 'touchdown()'. The following steps are followed:

1. If layer 1 is selected (see section 2.4.15), a for-loop runs 1000 times, incrementing by 1.
2. All members of integer array 'A/B/Cpanloop[]' are set to -5. This array is used to store x/y coordinates of the path. This action resets the coordinates.
3. Variables 'A/B/Citems', 'A/B/Cloopcount', 'A/B/Cstartframe', 'A/B/Cpan' are reset.
4. 'ofPath' path cleared. Variable 'A/B/Cloop' is set to false.
5. Secondary path paused (see section 2.4.15).

This effectively deletes any currently drawn layer 1 path, pauses all layers and resets required variables. Next the user will move their finger. 'Touchmoved()' in apolloapp.mm will record coordinate positions and relevant timings up to a maximum of 500 coordinates:

1. Only id=0 is allowed. This means accidental second finger touches are disregarded as paths can only be drawn in single touch.
2. Speed variable is reset to zero (see section 2.4.13.6).
3. X/Y coordinates are recorded into integer array 'A/B/Cpanloop[]' at index 'A/B/Cpan' (x) and 'A/B/Cpan+1' (y).
4. 'A/B/Citems' is updated with the current quantity of X/Y coordinates.
5. The start time is recorded into variable 'panloopstartframe' using the function 'AJseconds()' and 'ofGetElapsedTimeMicros()' (section 2.4.16).

6. The relevant time of the current x/y coordinate is recorded in array 'panlooptime[]' at index 'A/B/Cpan'/2.
7. An 'ofPath' object is used to draw the path. The dot operator 'lineto' is used to draw a line to the current point.
8. The path is made visible and variable 'a/b/cclear' set to false.
9. Variable 'A/B/Cpan' is incremented by 2.
10. If 'A/B/Cpan' reaches 1000, it is reset and the path is cleared. This is a safety mechanism that ensures that paths do not become too big. This can be expanded in the future if required.

The path is displayed using 'draw()' in 'apolloapp.mm' using the function 'AJ_PanLevelx()'. The function takes 1 integer argument (1=A, 2=B and 3=C). The first part of the function creates local variables for use within the function; the second initializes these variables using global variables; the third performs the main action of the function; the fourth part resets global variables. In this way, the same function can be used for different paths without having to re-write the main body of the function. The steps that are followed in the main body of the function are described in the following sections.

2.4.13.2 PLAYING

If the play button is pressed (layer 1), 'A/B/Cloop' is made true and 'unpauseinstance' is set to true (section 2.4.13.3). The path plays back with the same timings as it was drawn. Within the main part of 'AJ_PanLevelx()' the following actions are taken:

1. If the 'ofPath' object is visible and has not been cleared, draw the path. The path is drawn using the dot operator 'draw'.
2. **Loop Setup:** A for-loop is set up incrementing 2 between zero and the amount of items in the coordinate array, cycling through the 'looptime[]' array.
3. **Cycle reset:** if the variable 'startframe', subtracted from the current time, is bigger than the last coordinate time in array 'looptime[]', set 'startframe' to the current time. This is used to reset the path when it reaches the end of the cycle. The start frame is used to shift the current time back to the times used in array 'looptime[]' (referred to as shifted time).

4. **'Looptime[]' array cycle:** If the shifted time is bigger or equal to the value at the current index of 'looptime[]' and less than the next index 'looptime[]' value, carry out the following operations:
    a. If the path is visible, draw the current position of the sound. This is represented by a circle of radius 30 with the colour of the pan loop. It is drawn at the coordinates at the current index value.
    b. Save the current coordinates in variable array 'stopcirc[]'.
    c. Work out the gains of each speaker in relation to the sound source position.
    d. Send OSC gain messages to each speaker (section 2.4.19.2).

2.4.13.3 PAUSING

If the path has been paused, 'A/B/Cloop' is set to false and 'pauseinstance' is set to true. Within 'AJ_PanLevelx()':

1. Path is drawn as above.
2. For-loop set up as above.
3. 'Pauseinstance' is true:
    a. Record current value of shifted time into variable 'pauseframe'.
    b. Set 'pauseinstance' to false.
4. The path is not playing therefore a circle is drawn at 'stopcirc[]' (the last unpaused coordinates).

If the path is unpaused, 'unpauseinstance' is set to true. This sets the start frame to the variable 'pauseframe' subtracted from the current time. This is done so that the path starts from the same place that it was paused and continues playing normally.

2.4.13.4 REVERSE

If the reverse button is pressed, 'reverseA/B/C' is set to true and 'reverseA/B/Cinstance' is set to true. This augments the way that the for-loop is cycled, effectively reversing it. Currently, the path is cycled using the index set to half the counter value. Variable 'timereverser' is the path index and 'i' is the counter. The path index is half the counter so that it loops through every value of the 'looptime[]' array:

```
timereverser=i/2;
```

When in reverse mode, the path index is changed so that the counter loops in reverse:

```
timereverser=((layer1items-i)/2);
```

Variable 'layer1items' is the amount of coordinate points in the path.

2.4.13.5 SINGLE PLAY

If the single play button is pressed, 'A/B/Csingle' become true. Within 'AJ_PanLevelx()', the variable 'startframe' is prevented from being reset when the time runs past the variables saved in 'looptime[]'. This causes the path to play through and stop playing at the last point.

2.4.13.6 SPEED CHANGE

When the Draw/Speed/Touch button is set to Speed Mode, the speed of the path playback can be changed. This sets the variable 'drawbuttonsetting' to 1 and 'A/B/Cpanspeed' to true. The speed is manipulated by swiping the finger 'up' the screen (along the y-axis) to increase speed and 'down' to decrease speed.

The mode is controlled using the function 'AJspeed()' which is called in 'apolloapp.mm' in 'touchmoved()' when 'menuset' is equal to 2. Within 'AJspeed()', the following occurs:

1. Local variable 'speed' is created.
2. If 'A/B/Cpanspeed is true', 'speed' is set to global variable 'A/B/Cspeed' depending on which path is selected.
3. Determine whether movement is within layer 1 (see section 2.4.15).
4. If y-axis position is decreasing (moving 'up' the screen):
    a. If 'speed' is less than 1, add 0.01 to 'speed'.
    b. If 'speed' is more than 1, add 0.1 to 'speed'. This adjusts the sensitivity of the speed change.
5. If the y-axis position is increasing (moving 'down' the screen).
    a. If 'speed' is less than 1, subtract 0.01 from 'speed'.
    b. If 'speed' is more than 1, subtract 0.1 from 'speed'.
6. If 'speed' is less than -0.98, set it to -0.98 to stop the speed becoming too small.
7. Reset 'A/B/Cspeed' to updated variable 'speed'.

2.4.14 TOUCH GESTURES

When the Draw/Speed/Touch button is set to Touch Mode, the path can be manipulated using several touch gestures. The buttons sets the variable 'drawbuttonsetting' to 2.

2.4.14.1 FUNCTION: 'AJPINCH()'

This function is used primarily within 'AJtouchgestures()' (section 2.4.14.2) in steps 6a and 7a and in 'AJ3fingerrotate()' (section 2.4.14.3). The function takes 3 arguments: the id of the touch; whether it is a touch instance or movement; whether the centre needs to be recalculated. It updates 4 main global variables: 'pinchcentrex/y' - the coordinates of the pinch centre; 'pinchout' – a bool determining the direction of pinch movement; 'pinchdistance' – how far the pinch has moved.

The pinch coordinates are shown visually as a grey circle created in 'apolloapp.mm' in 'draw()' when a touch operation is being performed.

The function operates in the following way:

1. If the touch is moving (2nd argument 'touch'=false):
    a. Make sure 1st and 2nd touch have not been released and that the movement coordinates are not equal to zero.
    b. Determine if the pinch centre needs to be recalculated ('recalculatecentre'=true); the centre recalculation is required by the rotate operation but not the scale operation to make sure the gestures respond in a natural way. If true, update 'pinchcentrex/y' in the following way:
        i. For 2 fingers:

```
pinchcentrex=moveid0x+(moveid1x-moveid0x)/2;
pinchcentrey=moveid0y+(moveid1y-moveid0y)/2;
```

Note the pinch centre is half way between the two fingers. Variables 'moveid0x/y' and 'moveid1x/y' are set at the start of 'touchmoved()' in 'apolloapp.mm' using the movement coordinates.

ii. For 3 fingers ('id2up'=false, the third finger hasn't been lifted) the central point of all 3 fingers is determined:

```
pinchcentrex=pinchcentrex+(moveid2x-pinchcentrex)/2;
pinchcentrey=pinchcentrey+(moveid2y-pinchcentrey)/2;
```

Note that the old 'pinchcentrex/y', determined from the 2 finger pinch, is augmented with the 3rd finger coordinates.

c. The direction of the pinch movement is determined by comparing the old 'pinchdistance' with the current 'pinchdistance'. This is done using the Pythagorean theorem with the x/y move coordinates.

```
if (pinchdistance>=sqrt(pow((moveid0x-moveid1x), 2.0)+pow((moveid0y-
moveid1y), 2.0)))
```

i. If the old pinch distance is bigger, then the pinch is moving inwards so 'pinchout' becomes false. Otherwise 'pinchout' becomes true.

d. Finally, the new pinch distance is determined.

```
pinchdistance=sqrt(pow((moveid0x-moveid1x), 2.0)+pow((moveid0y-
moveid1y), 2.0));
```

2. If the touch is not moving, only a touch down has occurred, the same process is repeated. Instead of 'modeid0/1/2x/y', 'touchid0/1/2x/y is used. This is useful for updating the global variables as soon as a touchdown operation has occurred.

2.4.14.2 FUNCTION: 'AJTOUCHGESTURES()'

The function 'AJtouchgestures()' is implemented in 'touchmoved()' in 'apolloapp.mm'. It takes 2 arguments: 'theid' – the id of the touch; 'whichloop' – the path being manipulated (0-5, one variable per pan path and layer). Within layer 1, the function works as follows:

1. Local variables are created
2. 'ofPath' variables are cleared.
3. Depending on the 'whichloop' argument, local variables are initialized with global variables.
   This function carries out translation, scaling and rotation operations.
4. **Translation.** The function always carries out a translate operation based on the movement of the first finger (id=0).

   a. An 'ofPoint' object is defined as the difference between the old finger position and the current finger position ('athingy') using the dot operator 'set'.
   b. The 'ofPath' path is translated by 'ofPoint' using the dot operator 'translate'.

5. **Scale.** If the second finger (id=1) is moving and the first finger has not been lifted, a scale operation takes place.
   a. The function 'AJpinch()' is used (section 2.4.14.1). The pinch centre is not recalculated during the gesture to make movement natural.
   b. The path is translated to the origin from the pinch centre. The scale operation works by scaling every x and y value, therefore the object needs to be translated to the origin so that the object is scaled appropriately.
   c. Depending on the direction of the pinch, the path is increased or decreased by 3%. Scaling is stopped when the path is at 20% of its original size.
   d. The variable 'scale' is used to monitor the current scaling of the path.
   e. The path is translated back to the pinch position using the dot operator 'translate'.
6. **Rotate.** If all 3 fingers are down, a rotate operation is carried out.
   a. Once again, the path is moved to the origin centered around the 3 finger pinch centre. In this operation, the pinch centre coordinates are recalculated continuously as the gesture progresses allowing for a predictable rotation.
   b. A 3D vector variable, 'axis', is set to be the origin.
   c. The function 'AJ3fingerrotate()' (section 2.4.14.3) is used to find the rotation amount. Each finger controls the rotation, allowing for precise control.
   d. Using the dot operator 'rotate', the path is rotated by the rotation amount around the axis.
   e. The path is translated back to the pinch centre position.
7. After the specific operation is complete, the points in the path must be resampled to update the coordinates held in 'arrayA/B/Cpanloop[]' to the new position of the path.

a. First a polyline variable, 'sampled', is filled with the outline of manipulated 'ofPath', 'floop'.
b. The array 'A/B/Cpanloop[]' and 'A/B/Citems are cleared. Using 1000 for-loop iterations.
c. A for-loop is set up to run through the quantity of the polyline determined by the dot operator 'size()'. The array 'A/B/Cpanloop[]' is filled with x and y coordinates of the polyline

8. Local variables are returned to appropriate global variables.

2.4.14.3 FUNCTION: 'AJ3FINGERROTATE()'

This function is used by the rotation operation within 'AJtouchgestures()' and returns the rotation amount in degrees. It takes 1 argument: the id of the moving finger. It works in the following way:

1. Local variables are created and initialized using global variables.
2. A safety check, to prevent erroneous first cases, is run.
    a. 3 variables are set to true when the 3rd finger is lifted: 'id2upinstance0', 'id2upinstance1', 'id2upinstance2'.
    b. If any of these is true when 'AJ3fingerrotate()' is accessed, the safety feature makes sure that the previous move position is deleted.
    c. The relevant instance variable is then set to false.
    d. This prevents the rotation value to be inaccurate on the first instance.
3. 'AJpinch()' is carried out to determine current pinch coordinates.
4. The current angle is determined using the tangent rule shown in equation 2.1, the ratio between the opposite and adjacent sides of a triangle:

$$tanA = \frac{opposite}{adjacent} = \frac{sinA}{cosA}$$

Equation 2.1

```
angle=(180/PI)*atan2((fingerx-pinchcentrex),(fingery-pinchcentrey));
        if (angle<0) {
            angle=360+angle;
        }
```

Note, the C++ function 'atan2(y,x)' is used which returns the principal arc tangent of y/x in radians. The return was multiplied by 180/π to give the angle in degrees rather than radians. If the angle is given as less than zero (counter clockwise movement), 360 is added to make the angle positive.

5. The variable 'oldangle' (previous angle) is equated in the same way using the previous move positions.
6. The function returns the difference between the current angle and the previous angle.

2.4.15 Multiple Layers

The application allows for multiple path automation layers. Currently, one extra layer is available although this can easily be extended in the future. Switching to pan layer 2 allows the user to automate the process of translating pan layer 1 by drawing a translation path. In this way, complex sound trajectories can be built quickly. The layer is selected within the 'Display Menu' using the stepper in the top right of the screen labeled 'Pan Layer'.

Figure 2.19 shows three snapshots of stereo left-right path (red) being translated across the surround sound space using a secondary, vertical path (orange). Part of the overall movement of the sound is represented in figure 2.20.

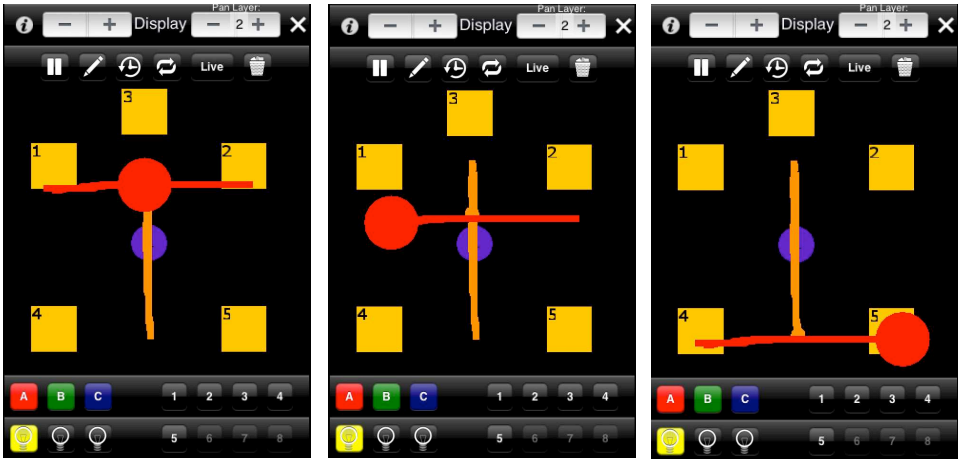


Figure 2.19 Snapshots of trajectory A playing on 2 layers

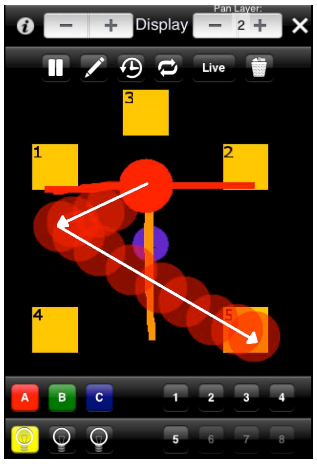


Figure 2.20 The combined movement of the sound for trajectory A playing on 2 layers

The second layer behaves in a similar way as the first. It is drawn by the user, can be played back at the same speed as drawing and can be manipulated in real time.

When the stepper is used to select layer 2:

1. The 'layerstepbutton' is operated within 'MenuGui.mm', updating the 'apolloapp.h' variable 'layer' using 'layerstep.value'.
2. This updates the layer number within 'MenuGui.mm' by setting the 'plabel':

```
plabel.text=[NSString stringWithFormat:@"%d", myApp->layer];
```

With Draw Mode set, the user can draw a layer 2 path. This translates the lower path and creates a new layer 2 path. When the user touches the screen:

1. Within 'apolloapp.mm' 'touchdown()', a for-loop runs which clears, resets and pauses the secondary path. This includes the coordinates, timing and item quantity.
2. The user moves their finger to draw a path. Within 'apolloapp.mm' 'touchmoved()', if the lower path has a path drawn ('items'>0), the function 'AJtouchgestures()' is run with variable 'whichloop' set to 0=A, 1=B or 2=C. The values for 'whichloop' correspond to the same values for layer 1 so that, as the layer 2 path is drawn and recorded, layer 1 path is translated.

Translating of layer 1 and drawing of layer 2 is handled by 'AJtouchgestures()' and 'AJpathrecord()'. Within 'apolloapp.h' in function 'AJtouchgestures()':

1. The translation of layer 1 occurs in the same way as step 4 of 'AJtouchgestures()' (section 2.4.14.2). Then function 'Ajpathrecord()' is called.

2.4.15.1 FUNCTION: AJPATHRECORD()

'Ajpathrecord()' takes 2 arguments: integer 'ABorC' (0, 1, 2) which sets the path selection and integer 'pathlevel' which specifies the layer.

1. Local variables are created and set depending on 'ABorC'.
2. The start frame is set:

```
AJseconds(ofGetElapsedTimeMicros());
```

   Functions 'AJseconds()' and 'ofGetElapsedTimeMicros()' are described in section 2.4.16.
3. The move coordinates 'moveid0x/y' are recorded into array 'PathTranslate[]'. The coordinate timings are recorded into array 'PathTranslateTime[]' at half the index value of 'PathTranslate' (one time value for every x/y coordinate). Note the timing is shifted with 'startframe' as with layer 1.
4. A temporary 'ofPoint' object is created and set to the current move position.
5. The main 'ofPath' object is then extended by using the dot operator 'lineto' to prolong the path to the current move position using the temporary 'ofPoint'.
6. Variable 'Clear' is set to 'false' to confirm that the path should be visible.
7. The counter and path items are incremented. The counter is used to set the index of the coordinate and timing array.
8. If the counter reaches 1000, all coordinates and timings are reset. This is to prevent the path of being too long. It can be extended in the future.
9. An instance variable is set to true. This is used during playback.
10. The last point is saved in a global variable to be used during playback.
11. Finally local variables are returned to global variables.

The layer 2 path is now displayed and recorded into the system. Now the path can be played back and manipulated.

2.4.15.2 PLAYING: FUNCTION 'AJPANLAYERXPLUS()'

Display and playback of the second layer is carried out by the function 'AJpanlayerxplus()' which is called by 'AJ_PanLevelx()' (also used to display and play back layer 1) in the local variable setup stage. The function takes 2 arguments:

'ABorC' to determine the path selection and an integer 'pathlevel' to determine the required layer.

In order to play the layer 2 path, the layer stepper must be set to layer 2 and the play button pressed. IBAction 'playbutton' in 'MenuGui.mm' is operated which turns the play button icon to a pause icon. As with layer 1, this operates function 'AJplaystop()' which sets variable 'A1/B1/C1play' to true.

Within 'AJpanlayerxplus()' drawing and playback works in the following way:

1. Local variables are created and initialized with global variables.
2. If the path is visible, not cleared and not in live mode, the 'ofPath' object is drawn using the 'draw' dot operator.
3. If the path contains more than zero items, a for-loop is run between 0 and the path items, incrementing by 2 each time using the counter 'i'.
    a. If this is the first playback since the draw operation, an instance variable has been set by step 9 of 'AJpathrecord()'.
        i. This creates a temporary ofPoint 'firstpoint' which is set to the first coordinate of the 'PathTranslate[]' array.
        ii. The layer 1 path is then translated to the origin using the ofPoint saved in step 10 of 'AJPathRecord()'.
        iii. The layer 1 path is then translated to 'firstpoint'. Translation is done using vectors. Coordinates are saved as absolute points. Therefore the path must be translated back to the origin before being translated to the relevant point otherwise the vector will add to the current position of the path.
        iv. The instance variable is set to false.
        v. A bool variable 'pathcount' (used as a reset flag) is set to 1.
    b. An index value 'timereverser' is set to i/2 (half the counter value)
    c. If the path is playing ('pathplay'=true):
        i. **Cycle Reset.** If the reset flag 'pathcount' is zero, the path is reset by setting the start frame to the current time.
            1. Flag 'pathcount' set to 1.
        ii. **Timing Array.** As with function 'AJ_PanLayerx()' for layer 1, an if-statement is used to check if the current time is between the current value of 'Pathtranslatetime[]' at index 'timereverser' or

the next time value. Shifted values of 'PathTranslatetime[]' are calculated by subtracting the start frame.

1. If true, the path is translated appropriately. This is done by creating 2 temporary ofPoint objects. The first acts as the translation vector to the 'origin' and the other acts as the translation to the next point ('tempa').
2. Variable 'origin' is set to the negative value of current position of the path by using index 'Path_pause_i' (step 3cii4).
3. For most cases, 'tempa' is set to the coordinates of 'PathTranslate[]' at index '(i+2)-x' and '(i+3)-y'.
4. The index variable 'Path_pause_i' is then recorded as 'i'.
5. However, when the path playback position is reaching the end of the path it is required to loop back to the start. Therefore, if 'i' is within 2 values (1 x/y coordinate) of the end of the path ('Pathitems'-2), 'tempa' is set to the *first* two items of 'PathTranslate' (i.e. the first coordinate). 'Path_pause_i' is 'reset' to -2 (in accordance with step 3cii4).
6. The layer 1 path is translated to 'origin' and then to 'tempa'.
7. A circle is drawn at the current position to mark the current position of the layer 2 path.
8. Finally, seeing as the layer 1 path has been translated, the coordinate points need to be resampled. This is done in the same way as in step 8 of 'AJtouchgestures()'.
    a. The outline of the ofPath of layer 1'lowerloop' is saved to an ofPolyline 'pathsampled'.
    b. A for-loop of size 1000 is run clearing the old values of the coordinate array.
    c. Another for-loop (using the quantity of items in the sampled ofPolyline) is run setting the coordinate array to the new values of the path.

2.4.15.3 PAUSING

Layer 1 and layer 2 playing and pausing is independent. Pausing the layer 2 path operates the Play/Pause button 'playbutton' in 'MenuGui.mm', and displays the 'pause' icon. 'AJplaystop()' sets global variable 'A1/B1/C1play' to false and sets variable A1/B1/C1pauseinstance to true. Step 3c in 'AJpanlayerxplus()' is bypassed because 'A1/B1/C1play' is no longer true. However the following additional actions take place within the function:

1. 'A1/B1/C1pauseinstance' is now true.
    a. Save the current shifted time in variable 'Pathpauseframe'.
    b. Set 'A1/B1/C1pauseinstance' to false.
2. 'Pathplay' is now false.
    a. Draw the position circle using array 'PathTranslate[]' at index integer 'Path_Pause_i'+1 and 'Path_Pause_i'+2 – this is the last coordinate position (see step 3cii5, section 2.4.15.2).

The path is now paused. Tapping the play button again follows the same instructions as in section 2.4.15.2 but also carries out additional actions. Operating the play button calls 'AJplaystop()' which now sets 'A1/B1/C1unpauseinstance' to true and 'Pathplay' to true. Within 'AJpanlayerxplus()', step 3c is active once more and the following additional actions are taken initially:

1. 'A1/B1/C1unpauseinstance' is now true.
    a. Set the 'PathTranslatestartframe' to the 'Pathpauseframe' subtracted from the current absolute time.
    b. Set 'A1/B1/C1unpauseinstance' to false.

The path is now unpaused and plays as normal from the paused position.

2.4.15.4 REVERSE

The reverse function works in exactly the same way as in layer 1. When the reverse button is operated in 'MenuGui.mm', 'reverseA1/B1/C1' is set to true. Within 'AJpanlayerxplus()' this changes the index variable 'timereverser' to (pathitems-i)/2, causing the index values to be read in reverse.

2.4.15.5 SINGLE PLAY

This mode is also very similar to layer 1. The 'singlebutton' within 'MenuGui.mm' sets 'A1/B1/C1single' to true. Within 'AJpanlayerxplus()', this prevents the cycle from resetting - step 3ci (section 2.4.15.2) causing the path to play once and stop.

2.4.15.6 SPEED MANIPULATION

When the draw/speed/touch button is set to speed mode, the user can swipe 'up' and 'down' the screen to manipulate the speed. Swiping up and down operates 'AJspeed()' (section 2.4.13.6), manipulating 'A1/B1/C1speed'. This changes the 'PathTranslate[]' array value in 'AJpanlayerxplus()' by dividing it by *1+'A1/B1/C1speed'*.

2.4.15.7 TOUCH GESTURES

Layer 2 touch gestures also use 'AJtouchgestures()' to carry out translation, scale and rotation operations (section 2.4.14.2). However the following additional steps are taken:

1. An additional ofPath variable 'lowerloop1' is set up and initialized as the layer 1 path.
2. The lower path is translated to the origin from its current position using array 'A1/B1/C1Translate[]' with index 'A1/B1/C1_pause_i'. A temporary ofPoint object is created with the coordinates and a translate dot operator is used on the layer 1 path to move it to the origin.
3. Translate and scale operations occur exactly the same as described in section: layer1 touch gestures.
4. Rotate occurs the same for the main path (in this case layer 2).
    a. A rotate operation is also applied to the lower path using the translate dot operator on the ofPath object 'lowerloop1'.
5. Resampling of points occurs in exactly the same way as described in section 2.4.14.2, step 7.
6. The lower path is translated back to the index 'A1/B1/C1_pause_i' of array 'A1/B1/C1Translate[]'.
7. The lower path is also resampled to update the coordinates.

2.4.15.8 LAYER 1 TIME DISPLAY

A simple time display is available for layer 1 paths accessed from the options menu (section 2.4.17). This gives the user a visual feedback as to the length of the path by displaying a white marker at each second. It also displays the time (in seconds) on the sound source itself to show how long the sound has been travelling on the current loop of the path.

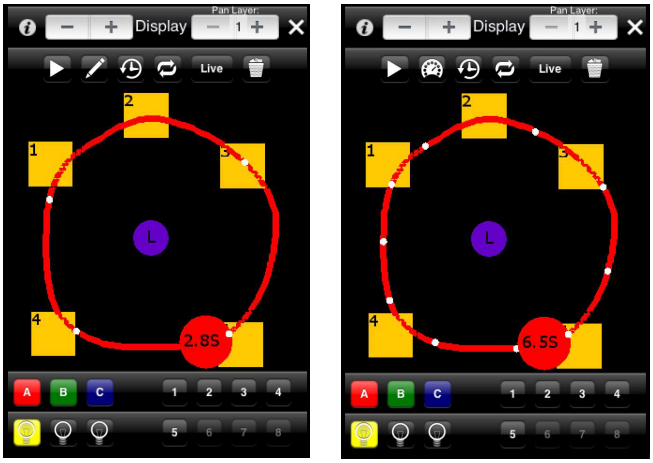


Figure 2.21 Time display markers showing a speed change

The left screenshot of figure 2.21 shows the second markers displayed on the path and the current time of the sound source on the source itself. The right screen shot of figure 2.21 shows that the user has slowed the speed of the path down, therefore more second markers are displayed and the time has increased on the source.

When time display is enabled, it makes variable 'timedisplay' true. Within function 'AJ_PanLevelx()' this adds the following additional steps:

1. **Second Markers**. Within the main for-loop, the time is saved in global variable 'temptime'.
2. A for-loop is setup that runs between the 'oldtemptime' and the current 'temptime'.
    a. If 'temptime' divided by 10 has a remainder of zero and the current 'temptime' is not identical to the 'oldtemptime' and is not zero, a white circle is drawn at that position. This is a second marker.
3. The 'oldtemptime' is set to the current 'temptime'.

The modulus function is used in step 2a. This outputs a remainder of division operation. The current 'temptime' and the 'oldtemptime' are compared so that multiple second markers are not displayed at the same second value.

To display the second within the sound source the following code is carried out whenever the main sound source circle is drawn:

```
ofSetColor(0, 0, 0);//set the colour to black
verdana14.drawString(ofToString((looptime[timereverser]/(1+speed))/100000.0,1)+
"S", panloop[i]-25, panloop[i+1]+5); //draw the string
```

This takes the current time of the sound source, divides it by 100000 so that it is in seconds and draws a string at the current sound source position within the circle.

2.4.16 TIMING

Timing is important in this prototype to make sure the paths behave in a predictable way. Originally, timing was carried out using 'frames' (i.e. the program cycle) which resulted in smooth operation. However, when the application was carrying out complex tasks, paths would run slower. Therefore timing in seconds was used instead. The function 'ofGetElapsedTimeMicros()' is used to acquire the time that the application has been running. The study refers to 'shifted time'. This is the current 'elapsed' time with a start time subtracted from it. It is used to retain relative times.

The function 'AJseconds()' is used to round values obtained from 'ofGetElapsedTimeMicros()' to a better format by using the modulus function.

2.4.17 OPTIONS MENU

When the user is within the Move, Display or Volume menu they can access the options menu by tapping the 'i' icon in the top left corner. This displays a translucent screen with various options as shown in figure 2.22.

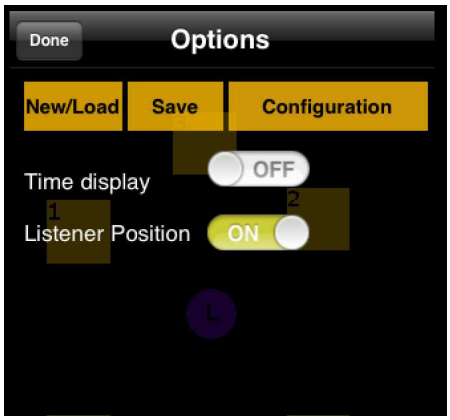


Figure 2.22 The Options Menu

The view is handled by the group of files 'OptionsView'. Clicking the 'i' button operates IBAction 'setupbutton' in 'MenuGui.mm' which sets the global variable 'optionsmenu' to true and hides the 'MenuGui' view. Within 'apolloapp.mm' in 'draw()', the view is 'unhidden'. The 'nomenu' GUI is hidden.

The Options View is made translucent by selecting the view in the 'xib' file and setting the alpha level of the background to 78%. If any paths are displayed, they will be visible while the options screen is activated so the user can keep track of activity. When the user has finished manipulating settings, they click 'Done' in the top left which returns them to the Move, Display or Volume menu. Clicking this button operates the IBAction 'backbutton' in 'OptionsView.mm' and sets the variable 'optionsmenu' to false which reverses the process by hiding the options menu and displaying the relevant main menu.

'New/Load' – Tapping this will go back to the title screen. It operates 'newbutton', setting the variable 'newscreeninstance' to true and 'optionsmenu' to false. The global variable 'newscreeninstance' is checked to be true using an if-loop in 'apolloapp.mm' in 'update()'. Here, it hides all views except 'workflow', which is made visible. This if-loop sets the titles of any saved files (see sections 2.4.6.2 and 2.4.18.2).

'Save' – This button saves the current settings of the project. It sets the global variable 'saveinstance' to true (see section 2.4.18.1).

'Configuration' – This opens the Configuration menu screen. The 'configbutton' is operated. This hides the options screen and sets global variable 'configoninstance' to true making 'MyGuiView' visible.

'Time Display' – A UISwitch is used to operate the time display. When the value is changed, it operates IBAction 'timedisplaybuttonswitch' which sets the global variable 'timedisplay' to the 'on' status of 'timedisplayswitch' using the dot operator 'on'.

'Listener Position' – This allows the user to turn listener position on and off (see section 2.4.19.2). The switch operates IBAction 'listenerbuttonswitch' which sets global variable 'listeneron' to the 'on' status of 'listenerswitch'. If the listener is switched off, the current listener coordinates (saved in array listenercoordinates[]) are saved into array listeneroffcoordinates[]. The listenercoordinates are then set to -1000 (offscreen). If the listener is switched on the listener coordinates are reset.

2.4.18 SAVING AND LOADING

The user can save and load project settings. Settings can be saved under a custom file name in the options menu. Up to 4 save slots are available. In the title screen (New/Load), the user can load one of the 4 files displayed by name.

When the user taps the save button in the options menu, the IBAction 'savebutton' is operated which makes global variable 'saveinstance' true. In 'update()' in 'apolloapp.mm', an if-statement monitors 'saveinstance'. When true, it displays a new view, 'savescreen' and makes the instance false. The view (figure 2.23) dims the background and shows a grey dialog box.

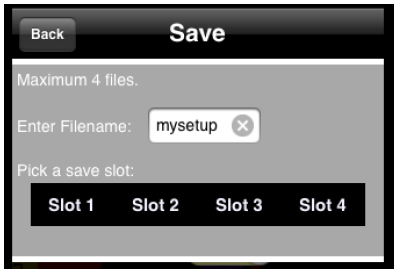


Figure 2.23 The Save Menu

The user must enter a file name and tap to pick a save slot. The status of the save operation is displayed at the bottom of the dialog box (not shown in figure 2.23). The user can also cancel the process by tapping the 'back' button.

The settings and filenames are saved in an Extensible Markup Language (.xml) file. A 'ofXmlSettings' object is created called 'AJXML' in 'apolloapp.h'. This is used to save, load and format xml files.

2.4.18.1 SAVING

When the user enters a name, and selects a slot, the 'status' is set to string 'Saving…', the number of the slot is saved to 'saveslot' and 'saveslotinstance' is set to true.

In 'apolloapp.mm' in 'update()' this triggers an if-statement that calls functions 'AJsavefilenames()' and 'AJsave()'. Variable 'saveslotinstance' is set to false and the 'status' variable is set to: 'Saved!'.

The function 'AJsavefilenames()' takes 2 arguments: the string 'latestfilename', which holds the user entered name and an integer 'slotnum'. The first argument is set as the text of 'savefield' located in 'savescreen' and accessed using the 'text' dot operator from the 'savescreencontroller'. The second argument is set using the variable 'saveslot', which depends on which slot button was pressed. Within 'AJsavefilenames()':

1. The array 'filenames[]' is used to hold the file names in the form of strings. Variable 'latestfilename' is saved at index "*'slotnum'-1*".
2. The 4 file names are saved into an XML file. Using the ofXmlSettings object 'AJXML', the file is formatted by setting the value of 4 tags (titled 'One', 'Two', 'Three', 'Four') to the values of 'filenames[]' at the appropriate index. The dot operator 'setValue' is used. The document created is saved in the following format:

```
<One>filenames[0]</One>
<Two>filenames[1]</Two>
<Three>filenames[2]</Three>
<Four>filenames[3]</Four>
```

3. The file is saved using AJXML and the dot operator 'savefile' with the directory as the argument. The directory is obtained in the following way:

```
ofxiPhoneGetDocumentsDirectory() + "filenames.xml"
```

'ofxiPhoneGetDocumentsDirectory()' returns the directory of the documents file associated with the application. The 'filenames.xml' string is appended to name the file.

Function 'AJsave()' is used to create and format the 4 slot files. It takes 2 arguments: the name of the file, in the form of a string, and the slot number. Information about the project is saved using tags to distinguish elements. The 'AJXML' ofXmlSettings

object is used to set values in a tag. E.g. the variable 'soundoutput' is recorded into the tags at a distinct location:

```
AJXML.setValue("DATA:SPEAKER:QUANTITY", soundoutput);
```

If 'soundoutput'=5, the format of the xml document will be:

```
<DATA>
        <SPEAKER>
                <QUANTITY>5</QUANTITY>
        </SPEAKER>
</DATA>
```

When saving large arrays, such as trajectory coordinate and timing information, it is useful to add a tag and then to 'push' into it using the dot operator 'addTag("TagName")' followed by 'pushTag("TagName")'. When finished, the 'popTag()' dot operator is used to 'pop' out of a tag. When saving large arrays, an extra argument is required when using the 'setValue' dot operator. This third argument specifies the position of the current value being saved. E.g 'i' specifies the position of the value using the counter:

```
for (int i=0; i<1000; i++) {

            AJXML.setValue("POSITIONS:A", Apanloop[i], i);
}
```

The function saves all relevant values and ends:

```
AJXML.saveFile( ofxiPhoneGetDocumentsDirectory()+"slot"+ofToString(slotnum)+".xml")
```

The 'saveFile' dot operator saves the file in the documents directory using the slot number specified by the user.

2.4.18.2 LOADING

Filenames are loaded when the application is started and when the 'Workflow1' view ('New/Load' screen) is shown. Initially the 'Workflow1.xib' file preloads the default names 'Slot 1/2/3/4' to the loading buttons.

During startup, within 'apolloapp.mm' in 'setup()', the function 'AJloadfilenames()' is called. This loads the 'filenames.xml' file using the 'loadFile' dot operator with the 'AJXML'ofXmlSettings object:

```
AJXML.loadFile(ofxiPhoneGetDocumentsDirectory() + "filenames.xml"
```

This is held within an if-statement to give feedback about the loading process. Next, the array 'filenames[]' is populated with the string values held in 'filenames.xml' by using the 'getValue' dot operator on 'AJXML' from tags 'One', 'Two', 'Three' and 'Four'. If no filename is stored in a slot, the name 'n/a' is used.

In order to load a project, the user must access the new/load screen. This is either done upon startup or by tapping the 'New/Load' button in the options menu. When viewing this screen, the user is presented with 4 buttons, loaded with filenames, representing the slots, under the 'Load' title (figure 2.24).

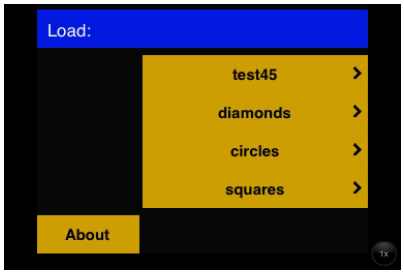


Figure 2.24 The Load Screen

Upon tapping one of the slots, within 'Workflow1.mm' the IBAction 'load0/1/2/3' is operated:

1. Function 'AJload()' is run with an argument specifying the slot number in the form of a string (e.g. 'slot1').
2. Variables 'menuset' and 'movemenu' are set to 1. This guides the application to the 'Move Menu' display.
3. Variable 'loadinstance' is set to true. This deals with the initial setup of the project after a load.
4. Variable 'menuvis' is set to 'false' to make the toolbars hidden.
5. Finally, the 'Workflow1' view itself is hidden.

The function 'AJload()' loads the variables from the XML file back into the application variables. It is simply the reversal of 'AJSave()' and works in a very similar way. The function takes 1 argument: the string filename relating to which slot number should be loaded.

1. The appropriate XML file is loaded from the documents directory depending on the argument.
2. Application variables are set from the stored values in the xml file using the dot operator 'getValue' on the 'AJXML' object. 'Pushing' into tags is used to speed up this process (see section 2.4.18.1).
3. OSC receiver and sender are setup.

The instance variable 'loadinstance' has been set to true. Therefore, within 'apolloapp.mm' in 'update()':

1. The Configuration menu view, controlled by 'myGuiViewController', is hidden.
2. Using the newly populated variable 'inputsources', if-statements check the quantity of input sources and enable the appropriate buttons: e.g.

```
if (inputsources>0) {
        MenuGuiController.source1.enabled=true;
        }
```

3. A for-loop runs through the input sources. Using the array 'inputsource[]', appropriate colouring (source routing) is applied to each active input source. First, a local UIColor variable 'tint' is set to the colour. e.g.

```
if (inputsource[input]==1) {
                tint=[UIColor colorWithRed:1.0 green:0.0 blue:0.0 alpha:0.0];
            }
```

This is assigned to the button tint. e.g.

```
if (input==0) {
                MenuGuiController.source1.tintColor=tint;
            }
```

2.4.19 OSC MESSAGE SENDING FUNCTIONS

The setup of the OSC sender and receiver are described in section 2.4.7. This section will deal with non-configuration messages sent out of the system to control panning and volume.

2.4.19.1 VOLUME

When the user is in the Volume menu (section 2.4.10), messages are sent to individual or multiple speakers as the volume is changed.

When the slider value is changed, function 'AJoscvol()' is called:

```
myApp->AJoscvol(i+1, myApp->speakervolume[i]);
```

Function 'AJoscvol()' takes 2 arguments: the speaker number and the volume. The values of these arguments are acquired as shown in section 2.4.10 and depend on speaker selection and which mode (relative or absolute) is selected.

The function works in the following way:

1. A temporary ofxOSCMessage 'message' is created.
2. A temporary string 'address' is created.
3. String 'address' is set to "/speaker/vol/".
4. The address of the message is set using the dot operator 'setAddress' on 'message' using 'address'.
5. An argument is added using the dot operator 'addIntArg' on 'message' using the 'speaker' variable provided in the function argument.
6. Another argument is added using the dot operator 'addFloatArg' on 'message' using the 'volume' variable provided in the function argument.
7. The OSC 'sender' then sends the message using dot operator 'sendMessage' with the 'message' as the argument.
8. The temporary variable 'message' is cleared using dot operator 'clear'.

The final OSC message is in the form: "/speaker/vol/ s v"; where s= speaker number (integer) and v= volume (float).

2.4.19.2 GAIN

When a sound source is routed to a path and that path is changing, either in live mode or in draw mode, OSC gain messages are sent using distance based amplitude panning (DBAP) (see section 1.3.4.4).

A for-loop runs between zero and the number of speakers. This operates the 'AJdbaposcgain()' function. This function takes 3 arguments: the pan path (1=A, 2=B, 3=C); the speaker number and gain. The value for the gain is obtained using function 'AJDBAPgain()'.

Function 'AJDBAPgain()' takes 1 argument: the speaker number and returns the gain as a float. It works in the following way:

1. Local variables are created and initialized.
2. If the listener is displayed, the listener gain factor is determined.

   a. The variable 'listenergain' is determined by using the sine power law (see section 1.3.4.1) to give a value between 0-1.
3. If the listener is not displayed, the variable 'listenergain' is set to 1.
4. A for-loop is run for each speaker.
   a. This finds the sum of the inverse squares of each speaker distance (section 1.3.4.4). Note, an spatial blur offset can be used.
5. The gain is found using the DBAP equation.
6. The gain is returned.

The function 'AJdbaposcgain()' now works in the following way:

1. A for-loop runs through every input source.
   a. Using variable array 'inputsource[]', if this matches the 'panpath' argument (i.e. the pan path is routed to the source):
      i. Follow steps the same way as in 'AJoscvol()'. However the address must be set to: '/speaker/sound/'.
      ii. The arguments are: speaker number, input number and the gain.
      iii. The message is sent and cleared.

This gives an OSC output in the form '/speaker/sound S I G'; where S= speaker number, I= input source and G= gain.

2.4.19.3 MUTE

When a pan path is de-routed or the mute button is pressed, a zero gain message must be sent to all the speakers at a particular source input. This is achieved using function 'AJzerogain()'. OSC message sending for the 'mute' button occur within function 'AJmute()'. Function 'AJzerogain()' takes 1 argument: input source and works in the following way:

1. A for-loop runs through the speakers.
   a. Messages are constructed in the same way as described in 'AJoscvol()'. With address: '/speaker/source/'. And with arguments: speaker, input source and gain (which is set to zero).
   b. The message is sent and cleared.

SECTION 3: EVALUATION AND DISCUSSION

3.1 TESTING TOOLS AND TECHNIQUES

3.1.1 MAX/MSP SPATIALISER

The prototype application is ready for integration with the Ensemble system. Further development by Apollo Creative is required, which is outside the scope of this study. Therefore, the behavior of the prototype application in manipulating sound and sending/receiving messages to/from an external computer was tested using a Max/MSP spatialiser. This handled: configuration message exchange; receiving and routing control messages; volume control; panning and sound output.

Ensemble's required output:

1. Control OSC message receiving/sending:
    a. Set IP address of the iOS device.
    b. Receive messages.
    c. Send appropriate configuration messages.
    d. Display messages for debugging.
2. Set the gain appropriately.
3. Set up test sound sources for panning.

The patch in 'presentation' mode is displayed in figure 3.1. Additional features are provided:

1. The quantity of speakers and sound inputs can be set.
2. iOS device IP addresses and port can be set.
3. The overall sound output can be toggled on/off.
4. The volume routing for the master volume and 3 sound source inputs are displayed in figure 3.1. These move automatically as OSC messages arrive into the system.
5. The input sources can be set to a constant sine tone, an intermittent sine tone and an audio file.

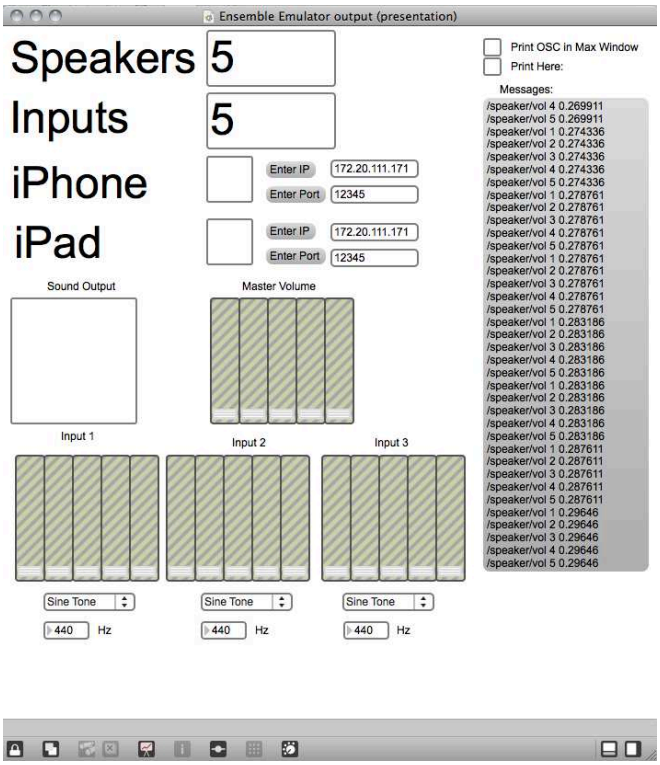


Figure 3.1 Spatialiser - Presentation

3.1.1.1 CONFIGURATION MESSAGES

During configuration (section 2.4.7.1), the destination IP is set and the device sends out a configuration message '/getconfig'. The destination IP, set in the application, is the address of the computer on which the spatialiser is running. The IP of the iOS device is entered in the patch.

The 'udpreceive' object is used to receive OSC messages. It uses the User Datagram Protocol (UDP) to receive messages sent over a network. For configuration messages, the subpatch 'setupsend' is used:

1. The 'sel' object is used to determine whether a configuration message is received.
    a. If '/getconfig' is the OSC message, a trigger object is used to send a bang to the user input speakers and input source quantity.
    b. A return OSC message is setup using the 'prepend' object to create a message in the form: '/config/speakerquantity S'; where S=speaker quantity; and '/config/soundsources I'; where I=sound sources.

c. Using the 'udpsend' object with IP and port set by the user, the message is sent out.

3.1.1.2 MASTER VOLUME

For the master volume OSC messages, the 'udpreceive' object is used again. The spatialiser allows a maximum of 5 speakers however this can be extended. Figure 3.2 shows the master volume output part of the patch.

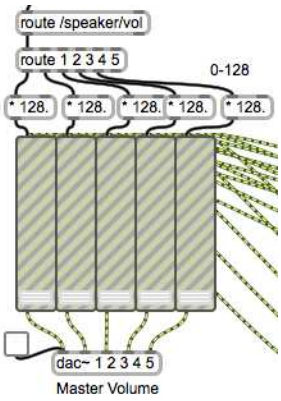


Figure 3.2 Spatialiser – Volume Control and DAC patch segment

1. A 'route' object is used to route messages that start with '/speaker/vol' and strip the address.
2. Another 'route' object takes the filtered message and routes it according to speaker number.
3. The output of the secondary 'route' object is multiplied by 128 (the range of Max/MSP's volume slider) and sent to the main volume slider for that speaker.
4. The 'dac' object (Digital-Audio Converter) is used to send out the final audio.

3.1.1.3 SOURCE ROUTING AND SOUND POSITIONING

For panning, gain messages specifying pan path and source are received by the spatialiser. Three stages of 'route' objects (Figure 3.3) are used corresponding to the 3 parts of the message.

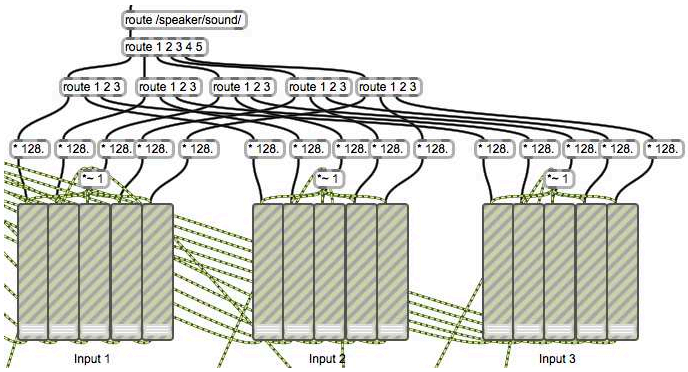


Figure 3.3 Spatialiser – 3 input panning patch segment

1. The first 'route' object selects and filters messages beginning with '/speaker/sound'.
2. The next object filters the messages depending on speaker.
3. The third object filters the messages depending on sound input routing.
4. Once again the gain is multiplied by 128 and sets the gain of the sliders.

3.1.1.4 SOUND SOURCES

The user can choose one of 3 sound sources from a drop-down 'umenu' (figure 3.4): a sine tone, intermittent tone or audio file. The output of the menu is routed to a 'sel' object which uses gates to operate the appropriate part of the patch.

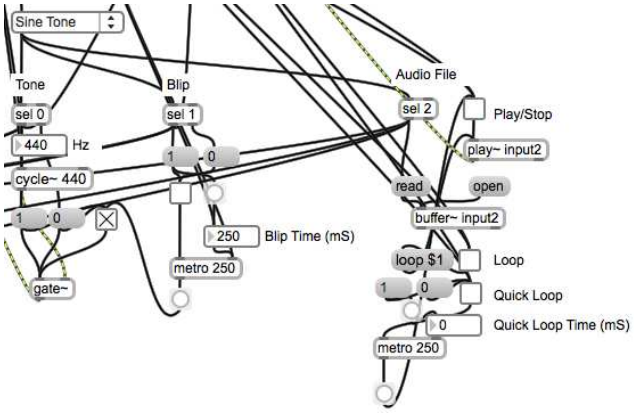


Figure 3.4 Spatialiser – Sound source selection

1. A 'cycle' object is used to play a sine wave at the user specified frequency.
2. The intermittent sine wave adds an extra step to this by using a 'metro' object to switch the sound on and off at the interval specified by a user.

3. The audio file option opens a dialog box allowing the user to pick a saved audio file. This is then loaded into a 'buffer~' object and can be played using the 'play~' object.

3.1.2 Testing Environment

Testing was carried out in the Live Music Production (LMP) room in the University of Huddersfield (Figure 3.5). Testing was carried out with up to 5 speakers and a range of positions. An Apple Macbook was used to run the Max/MSP patch. Messages were sent over the University of Huddersfield Wi-Fi network and an ad-hoc connection.

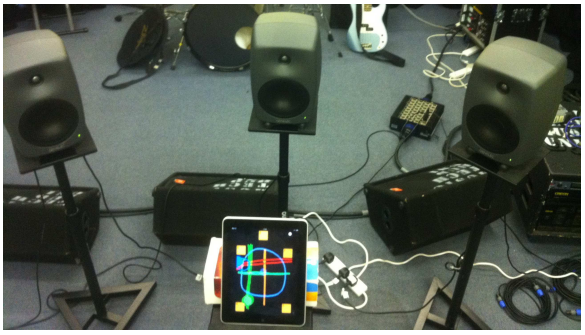


Figure 3.5 LMP Testing

3.1.3 Diagnostics - Methodology

Within Xcode, the static analyser can be used to investigate the source code looking for flaws in logic, memory management and unused variables (dead store) (Apple, 2012). When building an application, Xcode also provides feedback, in the form of warnings and errors, which can highlight major flaws that can stop the application running or predict run-time errors.

Apple has provided a set of diagnostic tools for discovering problems and analyzing the way the application works called 'Instruments'. The 'activity monitor' records CPU usage, memory and network activity (Apple, 2010). The activity monitor takes samples of the application activity every second. It charts this in a graph and provides information regarding the running statistics of the application.

3.1.3.1 Activity Monitor: Total Load

Apple Instruments 4.2 was used for testing. With the iOS device connected via cable, the following test method was followed:

1. The Apple 'Instruments' software is started.
2. The 'Activity Monitor' template is selected for the trace document.
3. The target is selected to be the study application: 'Apollo'.
4. 'Record Trace' is selected. This opens the application on the device and begins taking readings.
5. The activity monitor samples the total load of the application every second as it is being used. Flags can be inserted at any stage to mark significant events.
6. Once testing is complete, recording is stopped. This also stops the application.
7. The 'Activity Monitor' now displays a graph showing Total load (%) against time (Seconds).
8. 'Samples' can be displayed by selecting from the drop down menu. This gives a list of information (including 'Total Load') at each second sample. Flags are used determine when particular user interactions occurred.
9. Results were recorded as appropriate.

Testing all the iterations of available variables is impossible. Therefore appropriate testing variables were chosen to test how the application functions work normally and during heavy use. Testing predominantly focused on time-dependent features such as path playback and touch gestures.

An example 'trace' (for the application start test, section 3.1.3.2) is shown in figure 3.6. Note the flags on the top used to mark when the interaction occurred and correspond to 'spikes'. Average idle loading can be observed in this way.

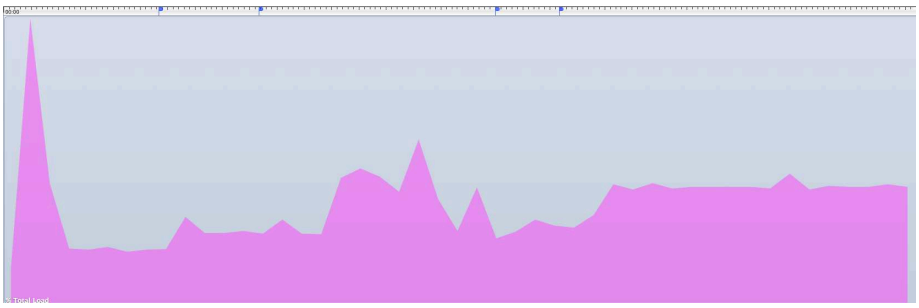


Figure 3.6 Application startup trace

3.1.3.2 APPLICATION START

The following readings were taken as the application was started and a new project was configured:

1. Application started.
2. New Ensemble File selected.
3. IP address entered.
4. 'Connect to Ensemble' button pressed.
5. 'Speaker Display' button pressed.

See section 3.2.1.1 for results.

3.1.3.3 SPEAKER MOVEMENT

The total load was measured as the speakers were repositioned. Several variables were taken into account:

- Distance: the speakers were moved horizontally and vertically to measure the full range of motion.
- Speed: the speakers were measured taking 1s and 5s to complete the movement.
- Amount of Speakers: the amount of speakers selected simultaneously via multi-touch was varied.

The screen translation operation was also measured with the variables: distance and speed. See section 3.2.2.1 for results.

3.1.3.4 PATH PLAYBACK

For path playback, the following variables were taken into account:

- The size of the path array. Two sizes were varied: a 'small' array of ≈100 coordinate points and a 'large' array of ≈900 points. It was difficult to use exact amount of points due to the sensitivity of the drawing feature.
- The speed taken to complete a path once. Two speeds of 1s and 10s were tested.
- The quantity of routed sound sources. The load was measured with 1, 4 and 8 sources routed simultaneously.

See section 3.2.5.1 for results.

3.1.3.5 TOUCH GESTURES

The total load of touch gestures was tested. The following factors were varied:

- The size of the path array: a 'small' array of ≈100 coordinate points and a 'large' array of ≈900 points.
- The speed it took to complete the gesture. Two speeds were used: 1s and 5s.
- For the 'move' or translate gesture, the two distances were measured: horizontal and vertical.
- For the 'rotate' gesture, 2 angles of rotation were measured: 90° and 360°.

See section 3.2.5.3 for results.

3.1.3.6 ADDITIONAL FEATURES

Volume mode: the amount of speakers selected (1-5) and the speed of volume change (1s and 5s) as well as the mute button were tested. See section 3.2.3.1 for results.

Saving and loading: The save feature was tested by taking readings when the 'save' option button was tapped, a file name was entered and a slot was picked. This was tested in a state with low and heavy background processes occurring. The low processes test consisted of 1 short path routed to 1 sound source. The heavy processes test consisted of 6 long paths and routing to 8 sound sources. Idle loading was recorded for the file recall test. See section 3.2.6.1 for results.

Reverse/Single play: two different pan paths (≈100 and ≈900 points) were tested. The idle state was also measured. See section 3.2.5.3 for results.

Drawing a path: two different paths were drawn: ≈100 points and ≈900 points. See section 3.2.5.1 for results. Note that the same size paths were used for all relevant tests.

Live mode: Live mode was tested varying distance (horizontal and vertical), routing (routed to 1, 4 and 8 sources) and speed (1s and 5s). See section 3.2.4.1 for results.

3.1.3.7 MESSAGE SENDING TIMING ACCURACY

The timing of messages received by the external system was tested. The Max/MSP spatialiser was extended to record messages arriving at the system and to measure the time at which they arrive. The time was measured using the 'cpuclock' object giving a

reading in milliseconds. The 'cpuclock' object was used in the 'cputimer' patch to measure the time since the last message. Figure 3.7 shows the patch segment.

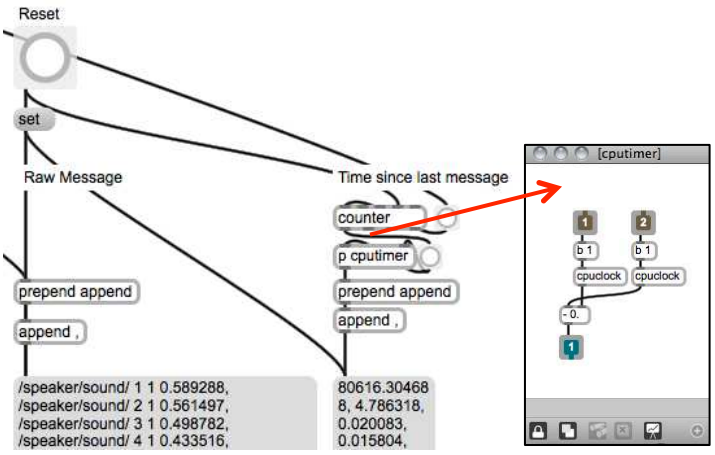


Figure 3.7 Spatialiser – message timing patch segment and cputimer subpatch

The raw messages received by the system and the time since the last message were recorded in two lists. The application code was modified to display the coordinate time in the debug window after the path was drawn using the 'printf' function.

With the iOS device connected, the test method:

1. A device-to-device connection was established between the computer and the iOS device.
2. The application was started using XCode. The debugging menu was active.
3. A new project was started in the application and it was navigated to the Draw Screen.
4. Path(s) drawn and routed.
5. The pan coordinate timings, displayed in the debugging window, are recorded.
6. The Max/MSP patch is reset.
7. The path is played through once, utilizing the 'single play' button.
8. Raw messages and CPU times are recorded into Max/MSP message boxes.
9. These are recorded into Microsoft Excel.

The experiment had the following variables:

a. Path length: ≈100 and ≈900 points were tested.
b. Routing: 1 and 8 sound sources.
c. Multiple paths playing: 1, 3 and 6 paths simultaneously.

3.2 DISCUSSION AND ANALYSIS

This section will discuss results of the testing process and interviews/questionnaires that were carried with Mark Hildred and Tim Anderson.

- Mark Hildred is the creative director of Apollo Creative, the developer of the Ensemble system and represents the current users of the system: teachers and music therapists. Mark Hildred had seen the application in development prior to the interview.
- Tim Anderson is a music technology consultant who has worked closely with the development of Ensemble and represents teachers who work with disabled people. Tim Anderson was seeing the application for the first time during the interview.

The interviews were guided by an evaluation questionnaire that provided a rating system. The following aspects of the features were rated from 1 (poor) to 5 (excellent):

- **Intuitive**: How easy the feature is to use?
- **User interface**: Is the design clear and aesthetically pleasing?
- **Usefulness**: Is the feature useful?
- **Functionality**: How well does the feature work?
- **Unique**: Have you seen features similar to this in other products?

Interviews focused on the usability of the application rather than software design principles. It was understood that the application was a prototype and a proof of concept rather than a commercial product.

3.2.1 APPLICATION START

3.2.1.1 TESTING RESULTS

At the application start, the total load jumps to 100%. This is due to the Openframeworks processes that occur when setting up the application and displaying the first window. Startup lasts on average 1.9 s.

Results are displayed in table 3.1. Configuration of a new project involves button presses, therefore the total load is represented as single values.

Event	Total Load (%)
Application Start	100
Idle	18.8
New file selection	30.2
Idle	24.6
IP address entering	57.4
Idle	22.6
Connect to Ensemble	29.2
Idle	27.1
Speaker Display	41.7
Idle	40.7

Table 3.1 – Application start - loading

The highest load during configuration was shown to be the entering of the IP address. This required several actions such as displaying/hiding the keyboard as well as setting the value of the IP itself.

3.2.1.2 INTERVIEWS

On the whole, the response from the interviewees was that the menu system's functionality was good enough with marks of 3 and 5 for the menu system as a whole. However there were many suggestions in terms of re-structuring the workflow of the application. This would change how initial configuration occurred.

3.2.2 SETUP: SPEAKER MOVEMENT

3.2.2.1 TESTING RESULTS

Figure 3.8 shows the loading as speaker movement occurs vertically/horizontally, at 1/5 seconds and with 1-5 speakers moving simultaneously. The whole screen translate is also shown. Full results are listed in appendix 5.3.1.

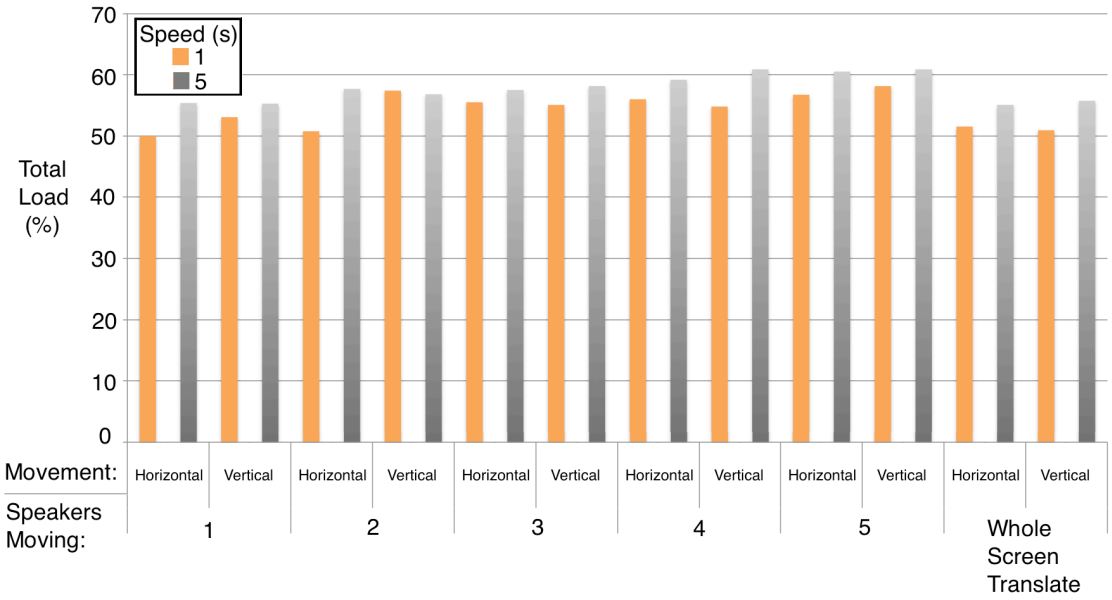


Figure 3.8 Speaker movement – CPU loading

Prolonged movement (5 s) of the speakers increased the load in almost every instance. As more speakers were held in multi-touch mode the total load increased up to a maximum of 60.9% with 4 and 5 speakers moving for 5s vertically. The 'whole screen translate' showed a lower total load because actual values of speaker positions were not affected.

3.2.2.2 INTERVIEWS

Table 3.2 shows how this feature was scored.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	4	Intuitive	4
UI	3	UI	3
Usefulness	4	Usefulness	4
Functionality	4	Functionality	4
Unique	3	Unique	5

Table 3.2 Speaker movement - questionnaire

The feature ranks highly for being intuitive, useful and functional. Tim Anderson noted that the feature was indeed very unique and Mark Hildred suggested adding default speaker positions (e.g. stereo, 5.0). The user interface was marked slightly lower and improvements included: units (distance and angle); making the 'window translate' feature clearer.

3.2.3 SETUP: VOLUME

3.2.3.1 TESTING RESULTS

Figure 3.9 shows the total load as the volume is manipulated depending on the amount of speakers, speed and if the 'mute' button is pressed. Full results are listed in appendix 5.3.2.

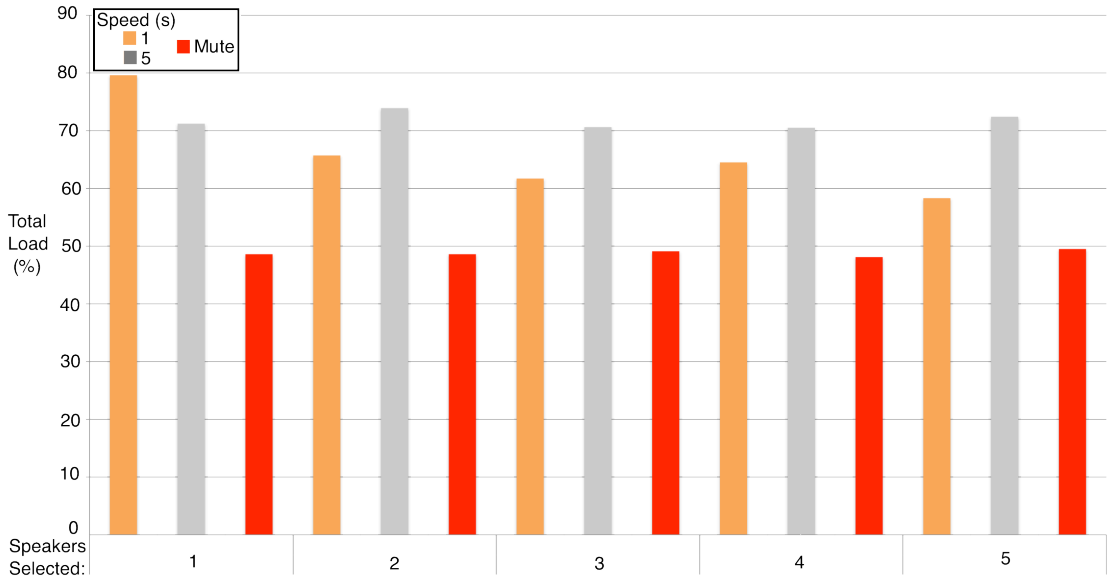


Figure 3.9 Volume – CPU loading

With regards to volume slider operation, the highest value came from 1 speaker, 1 second (79.6%) and the lowest from 5 speakers, 1 second (58.3%). Excluding this, loading was relatively stable on average at 62.6% (1 s), 71.7% (5 s). The mute button generated a constant load of 48/49%.

3.2.3.2 INTERVIEWS

This was a simple feature and was shown to work well and was regarded as useful as shown in table 3.3.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	3	Intuitive	4
UI	3	UI	3
Usefulness	4	Usefulness	4
Functionality	4	Functionality	3
Unique	3	Unique	4

Table 3.3 Volume - questionnaire

Suggestions included making the 'Relative' mode turn on automatically if speakers have a volume discrepancy. Speaker selection may require further user testing to determine whether the currently implemented 'shift-selection' and the 'Left/Right/Centre' buttons are appropriate as the interviewees differed in opinion.

3.2.4 PATHS: LIVE MODE

3.2.4.1 TESTING RESULTS

Live mode loading was tested, shown in figure 3.10, varying the distance, speed and amount of sources routed. Full results are listed in appendix 5.3.3.

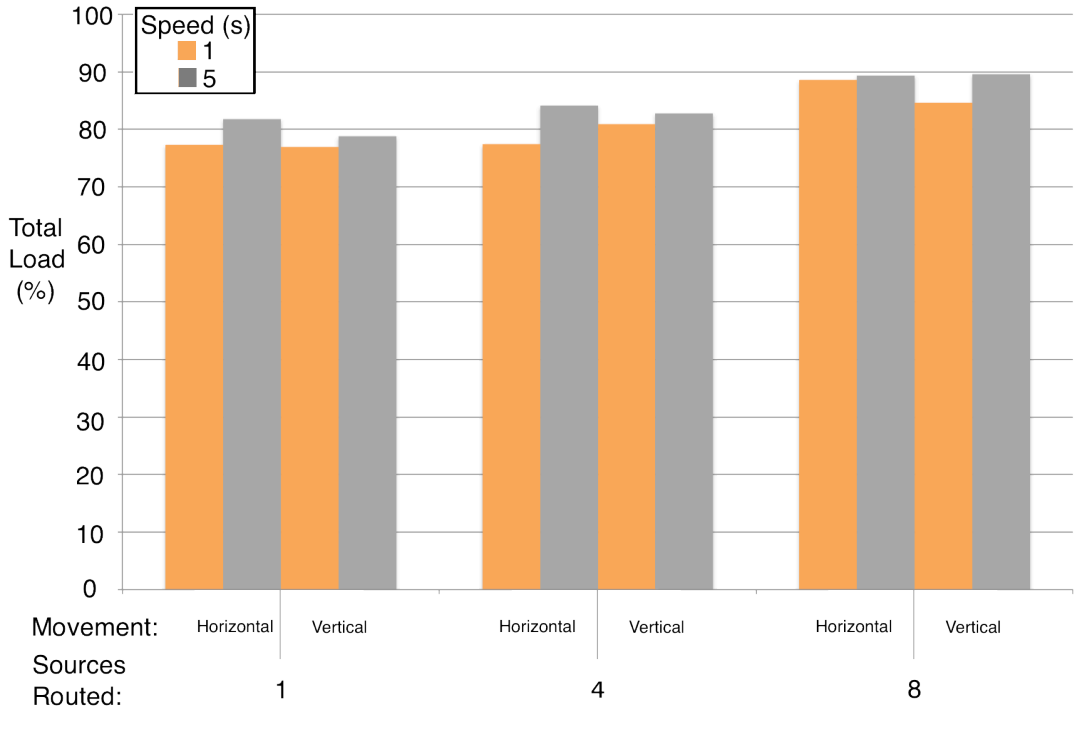


Figure 3.10 Live mode – CPU loading

As more sources are routed, the total load increases. The maximum load is 89.6% when the panner is moving vertically for 5s with 8 sources routed. There is an increase in total load if the speed is slower. This may be due to processing more points as opposed to a quick speed where the sensitivity of the touch interface processes fewer points.

3.2.4.2 INTERVIEWS

Live mode was seen by all the interviewees as a highly functional and useful feature as shown in Table 3.4. Surprisingly, it was seen as a unique feature.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	4	Intuitive	4
UI	4	UI	3
Usefulness	5	Usefulness	4
Functionality	4	Functionality	5
Unique	5	Unique	5

Table 3.4 Live mode - questionnaire

This feature was noted as being one of the most useful features for the primary user group as it is instantaneous and very simple to understand.

3.2.5 PATHS: DRAWING AND PLAYBACK

3.2.5.1 TESTING RESULTS: TOTAL LOAD

Drawing a path was tested with a large and a small path and the peak total load was measured. The results are shown in the table 3.5.

Path Length (points)	**Total Load (%)**
103	60.6
907	84.8

Table 3.5 Drawing a path – CPU loading

The idle total load was recorded as 40.7%. The load is higher when the path was larger because more points were being recorded and displayed.

For path playback, an experiment was run varying the amount of paths playing, the size of the paths, the speed of playback and the amount of routing as described in section 3.1.3.4. The size of the paths was: 103 points and 907 points for the small and the large path respectively.

The results are shown in figure 3.11. Full results are listed in appendix 5.3.4.

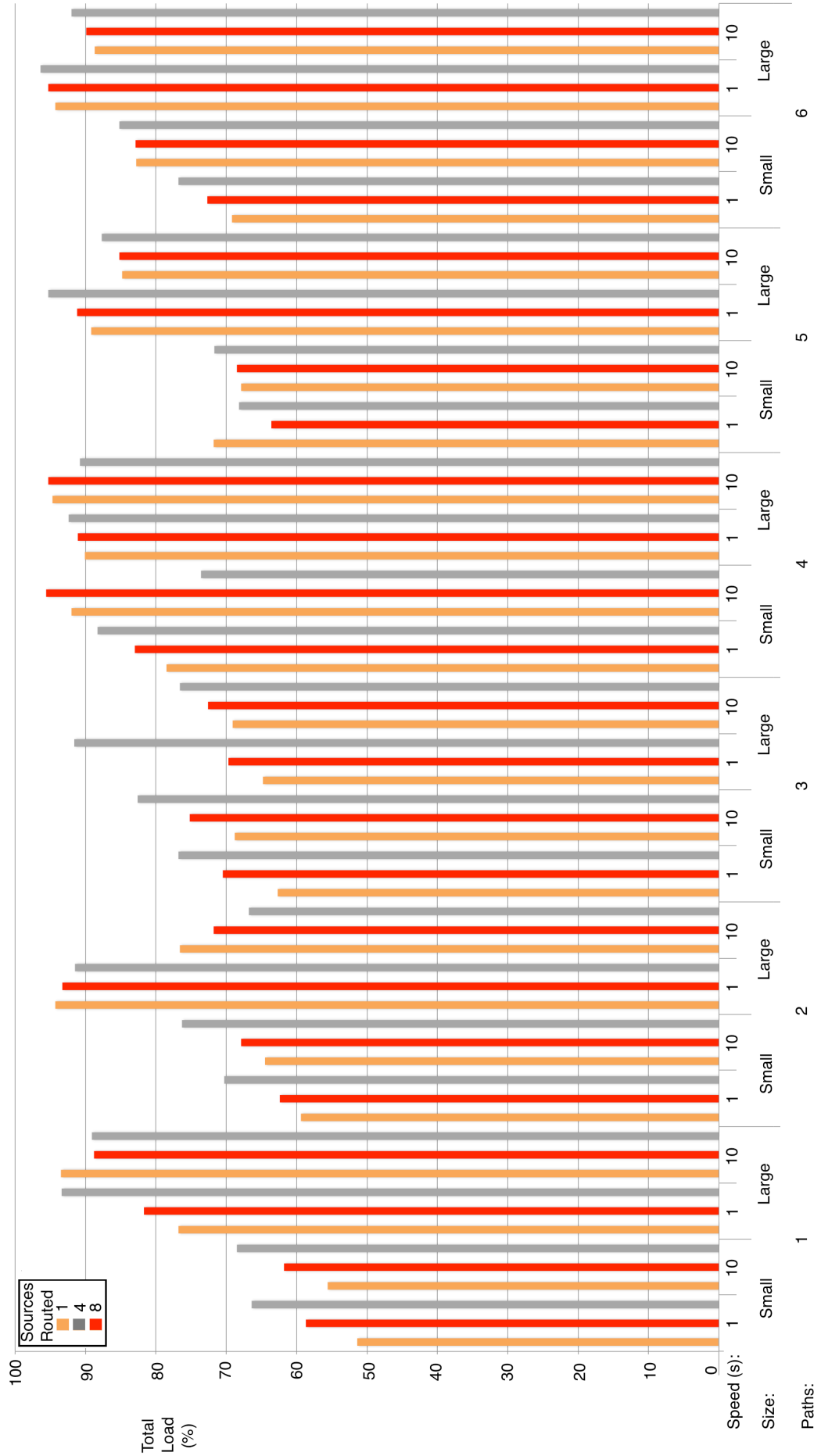


Figure 3.11 Path Playback – CPU loading

The biggest relative increase in loading depends on the size of the path. There tends to be an increase in loading as more sources are routed and when speed increases. The largest loading value was 96.4% - 6 large paths, routed to 8 sources, taking 1 second to complete. This is understandable as there is a lot of processing occurring visually, and in terms of message sending. During the test, it was noted that, visually, the movement of the paths started to become less smooth as the amount of paths, and their size, increased.

3.2.5.2 TESTING RESULTS: MESSAGE SENDING

The message sending of the system was tested. As described in section 3.1.3.7, a Max/MSP patch was developed to record message arriving into the system and the associated time. Initially, all the messages were recorded and displayed. However, when a large amount of messages were being received (>8 sources routed to a large path), Max/MSP stopped operating effectively. Therefore the testing patch was augmented to keep track of messages but only recording the cumulative time. Figures 3.12 and 3.13 show the cumulative time (i.e. the time it takes to receive all the messages into the system for one 'loop' of the path trajectory). A small (105 points taking 828mS – figure 3.12) and large path (975 points taking 8253mS – figure 3.13) were tested. The original path cumulative time is displayed for comparison. Full results are listed in appendix 5.3.5.

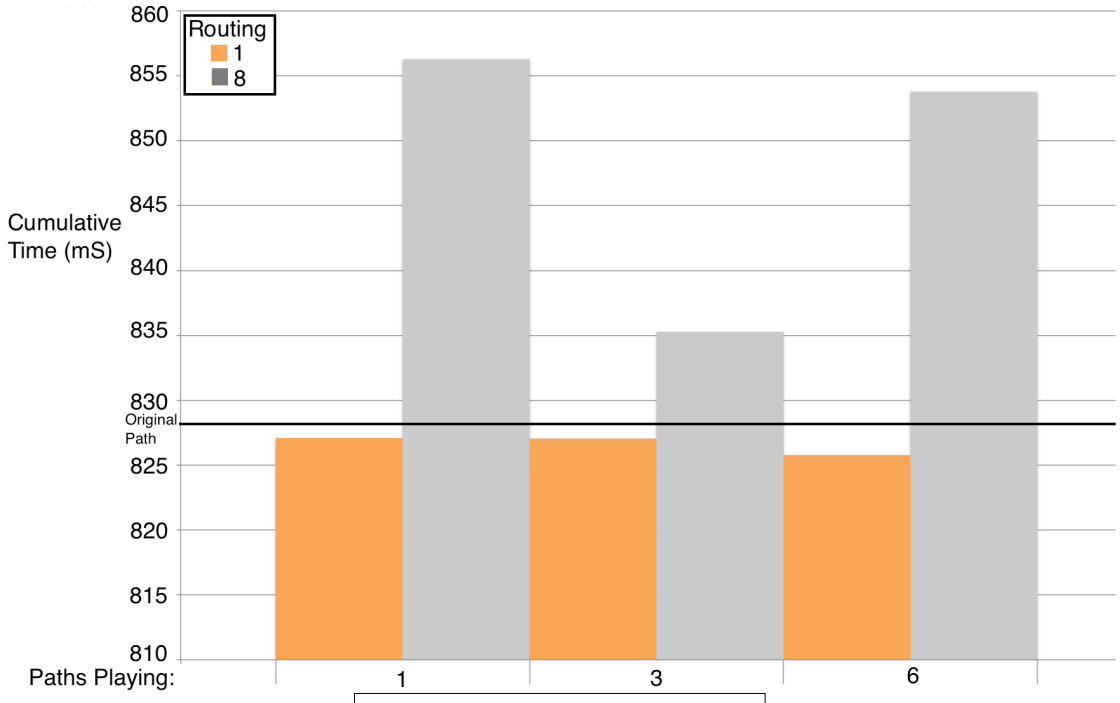


Figure 3.12 Message timing – Small Path

In figure 3.12 (small path), some results are less than the original path time. This result may be due to several factors including: the offset of the 1st message received into the system; timing differences between the device (send) clock and the receiver clock; dropped messages (although this is unlikely). The largest timing differences occur when the path is routed to 8 sources – more messages are received. The maximum difference between the original and received times is 28mS for 1 path playing, routed to 8 sources.

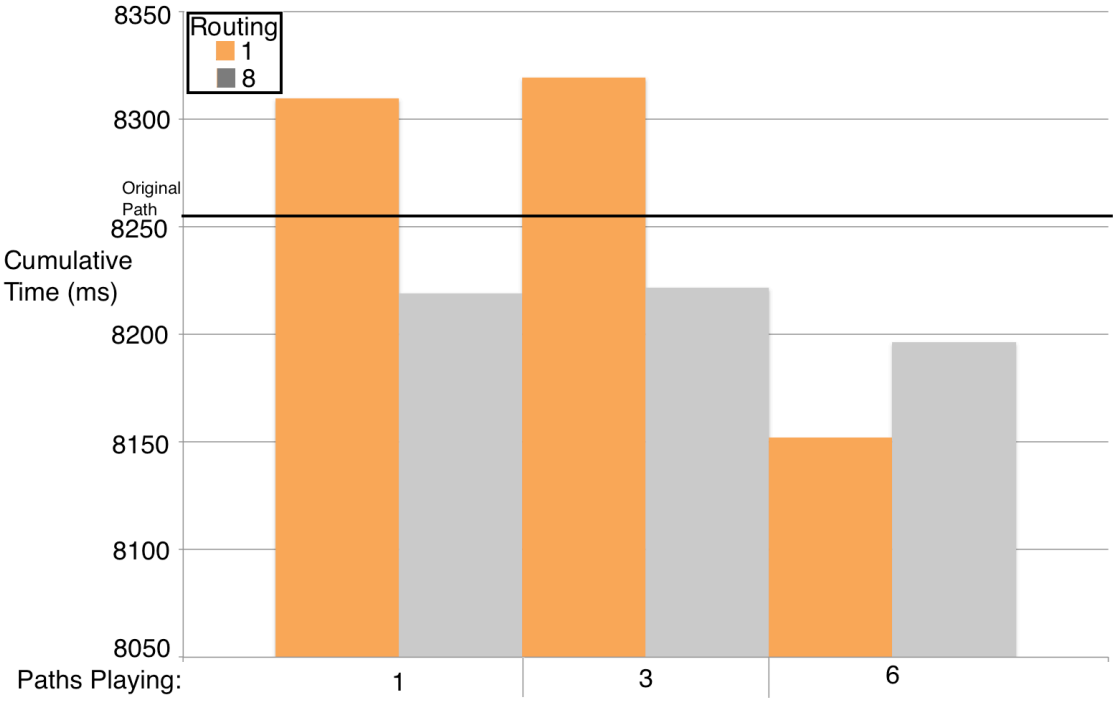


Figure 3.13 Message timing – Large Path

In figure 3.13 (large path), as with the small path, some results are less than the original path time. Here, the largest difference between original and received message time is 101mS for 6 paths playing, 1 source routed.

Studying the original experiment, where both messages and associated times were recorded it shows that messages are being sent correctly. There is a short delay between messages of the same coordinate point/time (around 0.02mS). This is due to processing speed and message sending. There is a larger delay between points as the system waits and processes data at the appropriate time.

3.2.5.3 TESTING RESULTS: MANIPULATION

The loading of single play, reverse buttons and touch gestures was measured.

The results of the single play and reverse button are represented as static loads shown in table 3.6. Idle loading was also measured for both paths.

Path Length (points)	Idle Loading (%)	Single Play button loading (%)	Reverse Button loading (%)
103	46.3	47.2	50
907	67.6	71.2	72.8

Table 3.6 Single and Reverse button – CPU loading

The loading of the single play and reverse buttons is low compared to the idle loading (2.3% higher on average) with the reverse button having slightly higher loading.

Separate experiments were run for the 3 touch gestures and the peak loading was recorded for each.

Loading when a path is moved in shown in figure 3.14. Full results are listed in appendix 5.3.6.

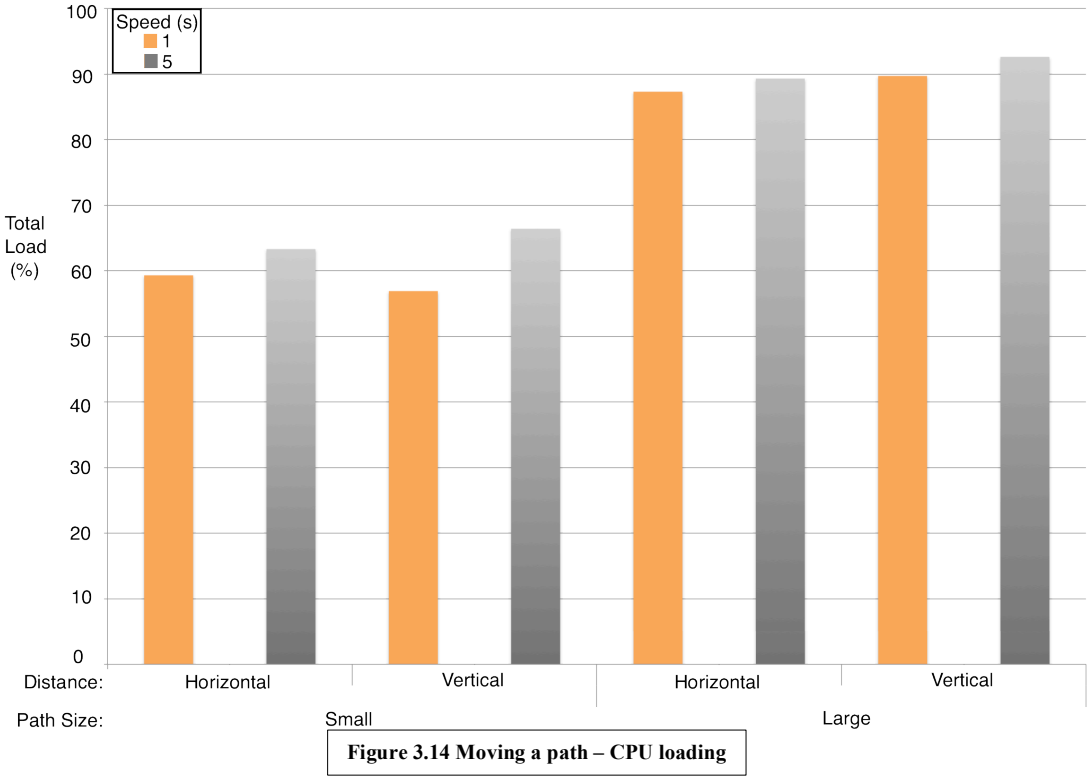


Figure 3.14 Moving a path – CPU loading

Load is dependent: on path size: increasing substantially for a large path; distance: overall, a larger, or vertical, distance shows a slight increase in load for a particular size; speed: a slower speed increases loading noticeably for a particular size and distance. The maximum total load is 92.6% for a large array of 907 points moving across the screen vertically in 5s.

Loading when a path is resized in shown in figure 3.15. Full results are listed in appendix 5.3.7.

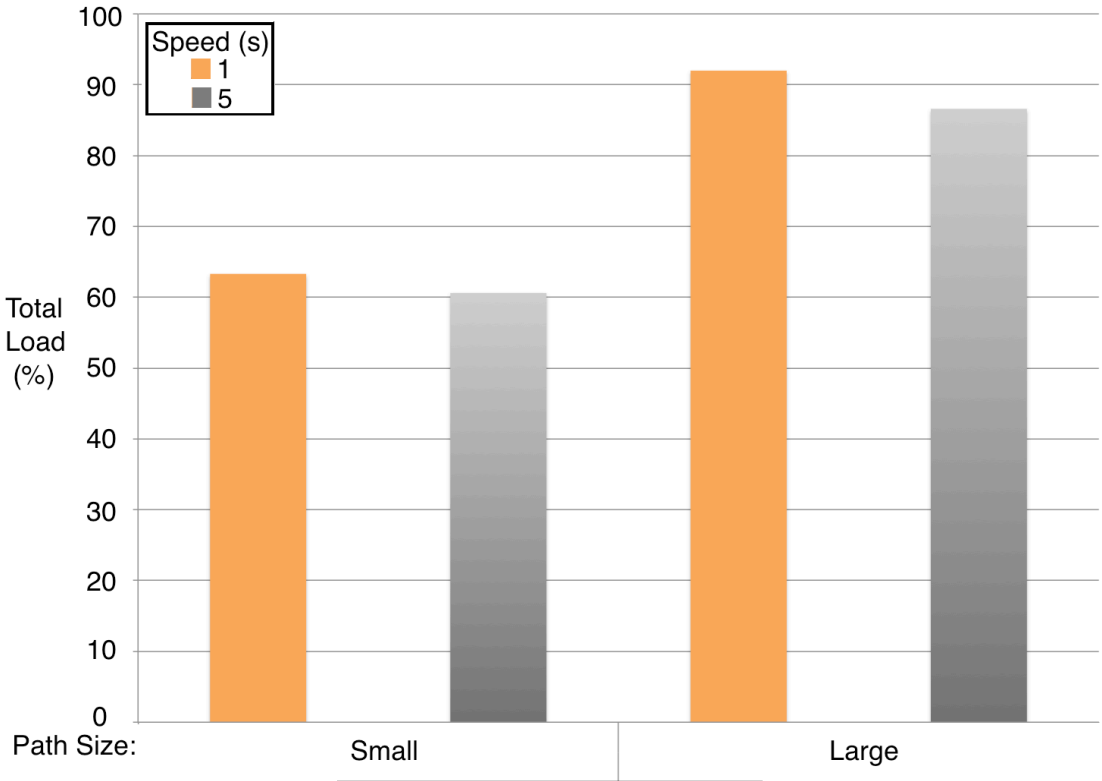


Figure 3.15 Resizing a path – CPU loading

The loading during resize increases if the speed is fast and if the array is large. The largest loading was 92% for a large path being resized for 1s.

Loading when a path is rotated in shown in figure 3.16. Full results are listed in appendix 5.3.8.

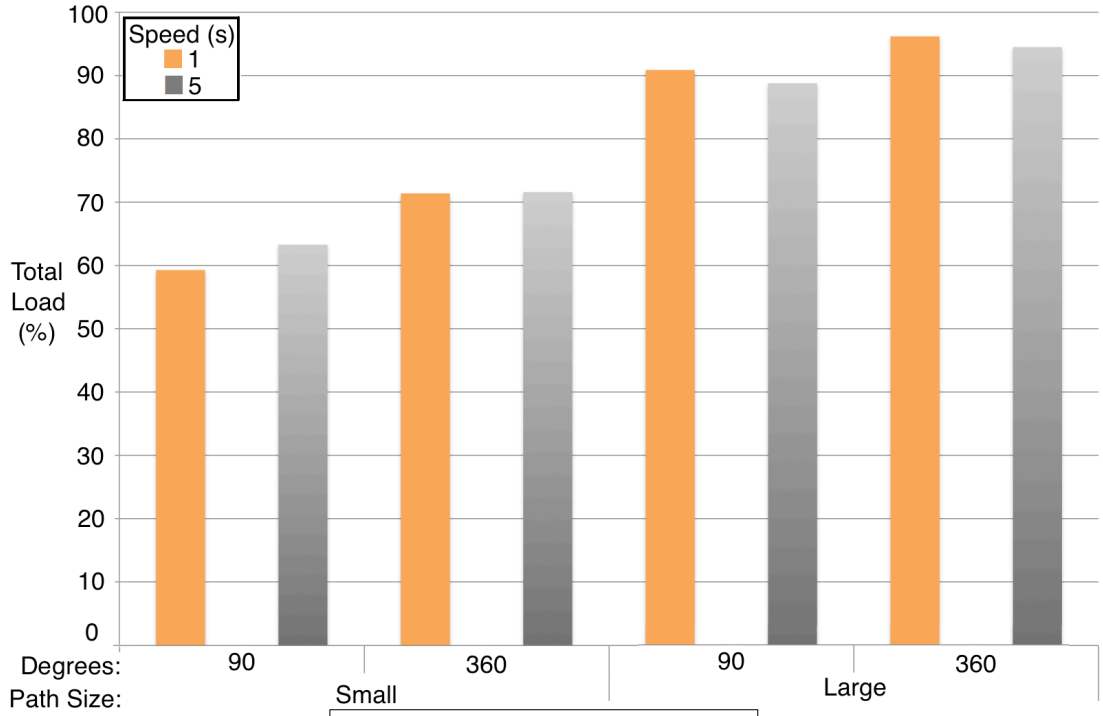


Figure 3.16 Rotating a path – CPU loading

The loading is affected by: the path size –a substantial increase between the large and the small path; the angle – loading increases significantly for the 360° angle each path size; the speed – loading tends to be higher when the speed is 1 s for a particular angle. The largest loading is 96.2% for a large path rotating 360° in 1s.

3.2.5.4 INTERVIEWS

Overall, drawing a path was seen as being easy to use, useful, very unique and modestly functional as shown in table 3.7.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	4	Intuitive	4
UI	3	UI	3
Usefulness	4	Usefulness	4
Functionality	3	Functionality	4
Unique	4	Unique	5

Table 3.7 Drawing a path - questionnaire

The drawing feature scored slightly less on UI. Path display was seen as relatively clear but it was suggested to increase the clarity of the source position and multiple on-screen paths. Mark Hildred recommended that the timing of the path could have the option of having a constant time rather than dependent on how it was drawn.

All interviewees stated that the prediction of movement and the overall level of control was good. The results are shown in table 3.8.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
On screen clarity	3	On screen clarity	4
Motion prediction	4	Motion prediction	4
Level of control	4	Level of control	4

Table 3.8 Path display - questionnaire

The feedback with regards to the application's handling of multiple layers is displayed in table 3.9.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	3	Intuitive	4
UI	3	UI	3
Usefulness	4	Usefulness	5
Functionality	4	Functionality	5
Unique	4	Unique	5

Table 3.9 Multiple layers - questionnaire

The response shows that, in terms of how useful, unique and functional the feature is, multiple layers performed well. However, when paths became numerous and complex, confusion could occur, lowering the UI score. Mark Hildred also noted that this feature was suited to more confident users, as it was a complex, high-level feature. It was also noted that the way in which layers were selected could be improved.

Feedback with regards to the single play and reverse buttons is displayed in table 3.10.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	2	Intuitive	3
UI	4	UI	3
Usefulness	4	Usefulness	5
Functionality	4	Functionality	5
Unique	3	Unique	5

Table 3.10 Single/Reverse buttons - questionnaire

It was noted that there was some icon confusion, lowering the score for how 'intuitive' the features are. However both buttons were noted as being very useful and functional.

The feedback for touch gestures is shown in table 3.11.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	4	Intuitive	4
UI	4	UI	3
Usefulness	5	Usefulness	5
Functionality	4	Functionality	4
Unique	5	Unique	5

Table 3.11 Touch gestures - questionnaire

This feature was highly praised for being very unique and incredibly useful as well as easy to use and functional. Ideas of how to expand the range of gestures to augment the path in various ways were suggested, including: point-by-point manipulation and shape skewing.

3.2.6 SAVING/LOADING

3.2.6.1 TESTING RESULTS

Testing was carried out to determine the CPU loading for saving and file loading. A low and high processing load was tested. Table 3.12 shows the results for the saving processes.

Events	**Low Processing Load (%)**	**Heavy Processing Load (%)**
Save Button (options menu)	62.6	92.6
Filename Entry	74.8	93.6
Slot button	100	100

Table 3.12 Saving – CPU loading

The application saves the settings accurately and does not exceed 100% total loading. However it is a feature that is processing heavy during the creation of the XML save file (when the slot button is pressed).

File loading was also tested. The results of this are shown in table 3.13.

Events	Low Processing Load (%)	Heavy Processing Load (%)
Load button pressed on startup	100	100
Post load idle	50.9	92

Table 3.13 File loading – CPU loading

Note that the processing during load for both files reaches 100%.

3.2.6.2 INTERVIEW

The feedback for the interviews for saving and loading is displayed in table 3.14.

Mark Hildred		**Tim Anderson**	
Feature	**Grade**	**Feature**	**Grade**
Intuitive	3	Intuitive	4
UI	3	UI	3
Usefulness	5	Usefulness	5
Functionality	4	Functionality	4
Unique	2	Unique	1

Table 3.14 Saving/Loading – questionnaire

The ability to save and load settings was deemed as a very useful tool that worked well. It is a tool that is almost universal to all software products therefore not unique. There was some suggestion as to improve this feature by displaying existing filenames during saving. Mark Hildred suggested expanding this feature to allow a user to save a specific room speaker configuration and volume without having to save paths (template saving).

3.2.7 FULFILLMENT OF SPECIFICATION

1. **'The application will run on iOS devices and send control messages wirelessly.'** – The application successfully sends Open Sound Control messages over Wi-Fi or an ad-hoc connection.
2. ***'The application will use simple colours and shapes.'*** – A small palette of colours was used. Pan trajectories were limited to 3 colours. Simple rectangles and circles were used to represent objects. There was a strong emphasis on providing visual feedback and colour coding related items.

3. ***'The user will be able to change the volume of the speakers individually and in a group. Control messages will be sent as the volume is changed.'*** – Volume mode was implemented successfully allowing single and group selection and volume change.
4. **'The user will be able to specify the position of the sound by directly touching and moving a finger (or appropriate stylus) on the touchscreen. This will send out control messages as the position is moved.'** – Live mode was implemented successfully. This provided 'instant gratification' and a quick transfer of ideas about sound position to reality without revealing the underlying, complex process.
5. ***'The user will be able to draw a sound trajectory onto the touchscreen. Once complete, it can be played back retaining the original timing. Playback will send control messages. The position of the sound will be shown on the screen.'*** – Draw and playback mode was implemented successfully. Drawing allowed ideas about sound movement to be quickly fulfilled in reality.
6. ***'The user will be able to route movement sensor inputs (sound sources) to the pan trajectories.'*** – Inputs can be routed to pan trajectories within the Display menu.
7. ***'The user will be able to reverse and change the speed of the sound trajectory as it is playing.'*** – A button was implemented to reverse the direction of the trajectory. The 'speed' mode was implemented that allowed the user to change the speed by swiping up and down.
8. ***'The sound trajectory can be played back in a continuous loop, during which no user interaction is required, or once.'*** – Within draw mode, the path could be played continuously. A button was implemented that put the path into 'single play' mode.
9. ***'The user will be able to use 1, 2 and 3 finger 'touch gestures' to respectively move, resize and rotate the sound trajectory as it is playing.'*** – Touch gestures were implemented to allow the user to move, resize and rotate the sound trajectory as it was playing. This allowed the user to modify the way the sound moves 'incrementally' and to safety experiment with ideas.
10. ***'The user will be able to draw multiple sound trajectory layers to move the trajectory below automatically.'*** – The application allowed 2 layers to be

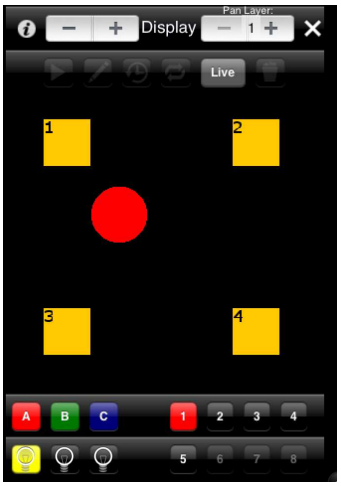
drawn. The primary layer (the first to be drawn, displaying sound position); the secondary layer translated the primary layer.

11. ***'The user will be able to position objects representing speakers arbitrarily on screen to represent the real-world speaker positions.'*** – Within Speaker Move menu, the speakers are represented by yellow squares and can be positioned on screen.
12. ***'The user will be able to save and recall application projects'*** – The save and load feature was implemented successfully.

3.2.8 Ensemble Integration

The application has been shown to function as required by the system specification. Currently, the application is ready for integration with Ensemble. However, further development is required by Apollo Creative to allow full communication and control.

An OSC input/output block for the designer software is currently under development by Apollo Creative. As well as receiving and sending messages, the block (or an additional block) will need to interpret messages and set speaker gains accordingly.



Control Messages:

/speaker/sound/ 1 1 0.764215

/speaker/sound/ 2 1 0.503336

/speaker/sound/ 3 1 0.579561

/speaker/sound/ 4 1 0.448743

Figure 3.17 4-Speaker example

Figure 3.17 shows control messages at the current position of the sound source. The Ensemble block would need to work in a similar way to the Max/MSP spatialiser used during testing. The volume of the individual speakers would be set to the pan gain multiplied by the volume. Figure 3.18 shows an example for speaker 1 (note the volume is 1) visualizing the process.

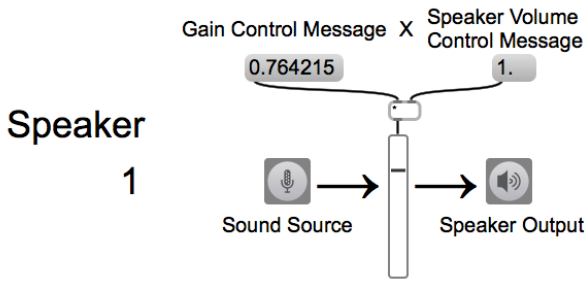


Figure 3.18 Proposed Ensemble block architecture

Currently, the Ensemble system deals with speaker gains by setting stereo pairs and assigning a pan between them. Alternative development could utilize this panning structure. However, this would require changing the panning equations to a pairwise panning method such as vector based amplitude panning or ambisonics instead of distance based amplitude panning. This would make sure the sound source travels on the speaker arc. Ultimately, this method would limit speaker positions.

3.2.9 Future Development

Ideas as to how to expand individual features have been discussed in the previous sections. Workflow was highlighted as an important direction to take in terms of future development. This involves guiding the user experience through the current set of features. Developing a suitable workflow would require significant user-guided testing which was not viable due to the time constraints of the study.

The prototype application already groups features to provide a simple workflow:

- Selecting a new project or loading a previous one.
- Configuration of the wireless settings.
- Positioning the speakers.
- Setting the volume.
- Controlling the sound by drawing and manipulating paths.

Currently, the application displays 3 example new project icons on startup which set up the project identically. Future development would tailor the initial conditions for different users. E.g. the primary user group would have easy access to only basic features such as live mode and path draw; secondary users would have the full range of features.

Mark Hildred suggested simplifying the amount of features available to the user at any one time by using two modes: design and play – similar to the current Ensemble workflow. This would move features such as speaker positioning, volume control, routing and (potentially) the drawing of paths to the configuration section. The 'play' mode would allow the users to position, play and manipulate paths without accidently overwriting previous paths. Arranging features in this way could make the 'play' environment less cluttered.

Other potential future developments include:

- Other panning and spatialisation algorithms such as VBAP and ambisonics.
    - Control messages can be expanded to include time delay, equalization and other spatialisation techniques.
    - Users can develop external spatialisation engines and use the raw x/y position data of the controller.
- Sound control can be expanded to allow control of stereo (or surround) sources.
- The controller has a versatile design that can be applied to other outputs:
    - Lighting control.
    - Additional multi-touch effects: delay/reverb, EQ manipulation.

3.3 CONCLUSION

This study set out to develop a proof of concept application focusing on the surround sound implementation of Apollo Creative's Ensemble audio-visual system. The primary users were specified as teachers and special needs specialists who are not necessarily technically proficient; a secondary group of advanced users were identified as composers and sound designers. The current products on the market focus on users that are familiar with complex music technology and are working in a professional environment with regards to their software framework, speaker placement and sound control.

An iOS application was developed. This functioned as a controller to move sound signals around an arbitrary, 2 dimensional speaker setup. Direct and automated sound positioning, in the form of pan trajectories, was implemented. The pan trajectories could be manipulated using buttons and touch gestures in real time. Open Sound Control messages were sent wirelessly and received by a Max/MSP testing spatialiser.

Testing and interviews were carried out to assess the operation of the application. Tests concluded that it operated successfully and efficiently within the remits of the device operating system. Overall, interviews deemed the features of the application unique, useful, functional and suitable for the primary and secondary user group. The study application fulfilled the original system specification.

Future development will focus on arranging features in a functional way to aid user workflow. Additionally, further panning algorithms, source control and non-pan sound manipulation should be investigated. Suitable user testing will be required to aid further development but this was unavailable due to time constraints of the project. Likewise, the application is ready for integration with Ensemble but requires some further development from Apollo Creative to allow full communication and control of the system.

The system was demonstrated and discussed at several conferences. A related paper was published in the proceedings of the International Computer Music Conference, Ljubljana, Slovenia 2012 (appendix 5.2). Please see appendix 5.1 for a summary of conferences and publications.

SECTION 4: REFERENCES

Aigen, K. (2005) *Being in music: foundations of Nordoff-Robbins music therapy*. Gilsum, NH: Barcelona Publishers.

American Music Therapy Association (2011) *History of Music Therapy* [online] Available at: <http://www.musictherapy.org/about/history/> [Accessed 5th January 2013].

Apollo Creative (n.d) *Apollo Ensemble* [online] Available at: <http://www.apolloensemble.co.uk/index.shtml> [Accessed 5th January 2013].

AppBC (n.d) *TouchAble* [online] Available at: <http://www.touch-able.com/touchAble/Features.html> [Accessed 9th January 2013].

Apple (2010) *iOS Human Interface Guidelines* [online] Available at: <https://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html> [Accessed 9th January 2013].

Apple (2010) *Performance Tools* [online] Available at: <https://developer.apple.com/library/ios/#documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html#//apple_ref/doc/uid/TP40001410-CH205-SW5> [Accessed 11th January 2013].

Apple (2011) *Logic Pro 9 User Manual* [online] Available at: <https://help.apple.com/logicpro/mac/9.1.6/en/logicpro/usermanual/Logic%20Pro%209%20User%20Manual%20(en).pdf> [Accessed 7th January 2013].

Apple (2012) *iPad – Technical Specifications* [online] Available at: <https://support.apple.com/kb/SP580> [Accessed 8th January 2013].

Apple (2012) *Performing Static Code Analysis* [online] Available at: <https://developer.apple.com/library/mac/#recipes/xcode_help-source_editor/Analyze/Analyze.html> [Accessed 11th January 2013].

Apple (2013) *iPhone 4 Tech Specs* [online] Available at: <https://www.apple.com/iphone/iphone-4/specs.html> [Accessed 8th January 2013].

Avid Technology (2011) *Pro Tools 10.0 Documentation*.

Bachmann, C. Bischoff, H. Bröer, M. Pfeifer, S. Schilling, H. (2010) *Cubase 6 Documentation*. Steinberg Media Technologies.

Barett, G. Omote, R. (2010) 'Projected-Capacitive Touch Technology'. *Touch Technology Issue*. 26, pp.17-18.

Baxter, H. Berghofer, J. MacEwan, L. Nelson, J. Peters, K. Roberts, P. (2007) *The individualized music therapy assessment profile – IMTAP*. London: Jessica Kingsley.

Benveniste, S. Jouvelot, P. Michel, R. (2008) 'Wii game technology for music therapy: A first experiment with children suffering from behavioral disorders'. In: *IADIS Game Conference*.

Blumlein, A. (1933) 'British Patent Specification 394,325'. *Journal of the Audio Engineering Society*. 6 (2), pp. 91-98.

Bonnet, J. (1715) *The History of Music and its Effects, from its Origins to its Present*. Paris.

British Association for Music Therapy (2012) *About BAMT* [online] Available at: <http://www.bamt.org/about-british-association-for-music-therapy.html> [Accessed 5th January 2013].

Bunt, L. (1994) *Music Therapy, An Art Beyond Words*. London: Routledge.

Bunt, L. Hoskyns, S. (2002) *The Handbook of Music Therapy*. London: Routledge.

Centre for New Music and Audio Technology (2012) *Introduction to OSC* [online] Available at: <http://opensoundcontrol.org/introduction-osc> [Accessed 5th January 2013]

CERN (2010) *Another of CERN's Many Inventions!* [online] Available at: <http://cds.cern.ch/record/1248908> [Accessed 8th January 2013].

Chomet, H. (1875) *The Influence of Music and Health and Life*. USA: G. P. Putnam's Sons.

Corke, M. (2002) *Approaches to Communication through Music*. London: David Fulton.

Davis, B. Thaut, M. Gfeller, K. (1998) *An Introduction to Music Therapy: Theory and Practice.* Dubuque: WCB.

Domestic Cat (2010) *Midi Touch* [online] Available at: <http://iosmidi.com/apps/midi-touch/ [Accessed 9th January 2013].

ELO Touch Solutions (n.d) *History of ELO* [online] Available at: <http://www.elotouch.com/AboutElo/History/> [Accessed 8th January 2013].

Erkkilä, J. (2007). 'Music Therapy Toolbox (MTTB) – An Improvisation Analysis Tool for Clinicians and Researchers'. In T. Wosch & T. Wigram (Eds.), *Microanalysis in Music Therapy*. London and Philadelphia: Jessica Kingsley.

Gerzon, M. (1973) 'Periphony: With Height-Sound Reproduction'. *Journal of the Audio Engineering Society*. 21 (1), pp. 2-10.

Gerzon, M. (1992) 'Panpot Laws for Multispeaker Stereo'. *Audio Engineering Society Convention*. March 1992, Vienna.

Gilboa, A. (2007) 'Testing the MAP: A graphic method for describing and analyzing music therapy sessions'. *The Arts in Psychotherapy*. 34 (4), pp. 309-320.

GitHub (2013) *hideyukisaito / ofxOsc* [online] Available at: <https://github.com/hideyukisaito/ofxOsc [Accessed 9th January 2013].

Hahna, N. Hadley, S. Miller, V. Bonaventura, M. (2012) 'Music technology usage in music therapy: A survey of practice'. The Arts in Psychotherapy. 39 (5), pp. 456–464.

Haque, A (2004) 'Psychology from Islamic Perspective: Contributions of Early Muslim Scholars and Challenges to Contemporary Muslim Psychologists'. *Journal of Religion and Health*. 43 (4), pp. 357-377.

Harrison, J. (1998) 'Sound, space, sculpture: some thoughts on the 'what', 'how' and 'why' of sound diffusion'. *Organised Sound*. 3 (2), pp. 117-127.

Hasselbring & Duffus (1981). Using microcomputer technology in music therapy for analyzing therapist and client behaviour'. *Journal of Music Therapy*. 18, pp. 156–165.

Heinrich, P. (2012) *The iPad as a tool for education*, Naace Research Papers.

Hexler (2013) *TouchOSC Documentation* [online] Available at: <http://hexler.net/docs/touchosc [Accessed 8th January 2013].

Holman, T (2000) *5.1 Surround Sound: Up and Running*. USA: Focal Press.

Huber, D. (2007) *The MIDI Manual – A Practical Guide to MIDI in the Project Studio*. USA: Focal Press.

Hunt, A. Ross, K. (2003) 'MidiGrid: Past, Present and Future'. In: *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME)*. Montreal, Canada.

Hunt, A. Kirk, R, Abbotson, M. Abbotson, R. (2000) 'Music therapy and electronic technology'. In: *Proceedings from EuroMicro conference*. 2, pp. 362–367.

Hunt, A. Ross, K. Matt, N. (2004) *Multiple Media Interface for Music Therapy*. IEEE Multimedia.

Jameson Proctor (2012) MxNM LE [online] Available at: <https://itunes.apple.com/us/app/mxnm-le/id364902749?mt=8> [Accessed 8th January 2013].

Jidkov, A. (2011) *Music Therapy Software Tool: An Interactive, motion controlled tool to aid non-verbal patient communication using behavior analysis*, University of Huddersfield, UK.

Jidkov, A. Gibson, I. Hildred, M. (2012) 'A Wireless, Real-Time Controller for the Ensemble Audio-Visual System'. In: *Proceedings of the International Computer Music Conference 2012*, IRZU_ Institute for Sonic Arts Research, Ljubeljana, Slovenia, pp. 273-276.

Johnson, E. (1965) 'Touch Display – A Novel Input/Output Device for Computers'. *Electronic Letters*. 1 (8), October 1965, p. 220.

Jorda, E. (2008) 'Music Therapy in Oncology'. *Clinical and Translational Oncology*. 2 (12), pp. 774-776.

Kostadinov, D. Reiss, J. Mladenov, V. (2010) 'Evaluation of Distance Based Amplitude Panning for Spatial Audio'. In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. USA.

Lee, C. (2000) 'A method of analyzing improvisations in music therapy'. *Journal of Music Therapy*. 37 (2), pp. 147-167.

Leeds College of Music (2012) *Leeds International Festival for Innovations in Music Production and Composition - Thursday & Friday 26 – 27 April 2012* [online] Available at: <http://www.lcm.ac.uk/latest-news/leeds-international-festival-for-innovations-in-music-production-and-composition> [Accessed 12th January 2013].

Liine (n.d) *Lemur User Guide* [online] Available at: <http://liine.net/assets/public_files/lemur/Lemur-User-Guide-3.1.0.pdf> [Accessed 8th January 2013].

Lorex (2012) *How Touch Screen Monitors Work* [online] Available at: <http://www.lorextechnology.com/support/self-serve/How+Touch+Screen+Monitors+Work/3100030> [Accessed 8th January 2013].

Lossius, T. Théo de la Hogue (2009) 'DBAP – Distance-Based Amplitude Panning'. In: *Proceeding of the International Computer Music Conference 2009 (ICMC).* Montreal, Canada.

Magee, W. & Burland, K. (2009) 'An Exploratory Study of the use of Electronic Music Technologies in Clinical Music Therapy'. *Nordic Journal of Music Therapy*. 17 (2), pp. 124-141.

Merging Technologies (2011) *Pyramix: Digital Audio Workstation – User Manual* [online] Available at: <http://www.merging.com/uploads/assets/Merging_pdfs/Pyramix_7_1/Pyramix%20User%20Manual.pdf> [Accessed 7th January 2013].

MIDIcreator (n.d) *MidiCreator – Music from Movement* [online] Available at: <http://www.midicreator-resources.co.uk/> [Accessed 7th January 2013].

MIDI Manufacturers Association (2012) *History of MIDI* [online] Available at: <http://www.midi.org/aboutmidi/tut_history.php> [Accessed 7th January 2013].

NewKinetix (2011) *Re Universal Remote Control User Manual* [online] Available at: <http://www.newkinetix.com/files/re-manual.pdf> [Accessed 8th January 2013].

Noble, J. (2012) *Programming Interactivity. USA:* O'Reilly.

Oldfield, A. (2006) *Interactive Music Therapy – A Positive Approach*. London: Jessica Kingsley.

Openframeworks (2012) *OF Start Up Guide* [online] Available at: <http://wiki.openframeworks.cc/index.php?title=OF_Start_Up_Guide> [Accessed 9th January 2013].

Pink Twins (n.d.) *Fantastick* [online] Available at: <http://pinktwins.com/fantastick/> [Accessed 9th January 2013].

Pioneer Corporation (2013) *iControl AV: Application Support & Guide* [online] Available at: <http://pioneer.jp/product/soft/appli/en.html> [Accessed 8th January 2013].

Pulkki, V. (1997) 'Virtual Sound Source Positioning Using Vector Based Amplitude Panning'. *Journal of the Audio Engineering Society, June 1997. 46 (6), pp. 456-466.*

Ross, K. Hunt, A. Hildred, M. Neighbour, M. North, F. (2002) 'Electronic Music Instruments – a role in Music Therapy?'. *Dialogue and Debate: Music Therapy in the 21st Century: A Contemporary Force for Change*. MusicTherapyWorld, Witten.

Rumsey, F. (2001) *Spatial Audio*. Oxford: Focal Press.

Saffer, D. (2009) *Designing Gestural Interfaces*. Canada: O'Reilly Media.

Saitara Software (n.d) *Ac-7 Core HD User Guide V1.01* [online] Available at: <http://saitarasoftware.com/temp/Welcome_files/Ac-7CoreHDv1.01.pdf> [Accessed 8th January 2013].

Shneiderman, B. (1983) 'Direct Manipulation: A Step Beyond Programming Languages'. *IEEE Computer Society Press*. 16 (8), pp. 57-69.

Skoogmusic (2012) *What is a Skoog?* [online] Available at: <http://www.skoogmusic.com/skoog> [Accessed 5th January 2013].

Sonic Arts Forum (2012) *Sonic Arts Forum* [online] Available at: <http://www.sonicartsforum.org.uk/> [Accessed 12th January 2013].

Soundbeam Project (2012) *Soundbeam: the Invisible, Expanding Keyboard in Space* [online] Available at: <http://www.soundbeam.co.uk/> [Accessed 5th January 2013].

Streeter, E. (2010) *Computer Aided Music Therapy Evaluation – Investigating and Testing the Music Therapy Logbook Prototype 1 System,* University of York, UK.

Thomas, P. (2013) 'Assist & Adapt: Music Technology and Special Needs: Part 1'. *Sound on Sound Magazine*. pp. 140-147.

Tidwell, J. (2006) *Designing Interfaces*. USA: O'Reilly.

University of Birmingham (2013) Meet BEAST Birmingham ElectroAcoustic Sound Theatre [online] Available at: <http://www.birmingham.ac.uk/facilities/BEAST/about/meet-beast.aspx>[Accessed 8th January 2013].

University of Huddersfield (n.d) HISS: Huddersfield Immersive Sounds System [online] Available at: <http://www.thehiss.org/> [Accessed 8th January 2013].

Wigram, T. Saperston, B. West, R. (1995) *The Art and Science of Music Therapy: a Handbook*. UK: Routeledge.

Wright, M. (1997) 'Open Sound Control: A New Protocol for Controlling Sound Synthesizers'. In: *International Computer Music Conference*. pp. 101-104.

SECTION 5: APPENDIX

CONTENTS

5.1 Conferences and Publications	133
5.2 ICMC 2012 Publication	134
5.3 Test Results	138
5.3.1 Speaker Movement	138
5.3.2 Volume	139
5.3.3 Live Mode	139
5.3.4 Playback	140
5.3.5 Message Timing	141
5.3.6 Move	142
5.3.7 Resize	142
5.3.8 Rotate	142
5.4 User Manual	143

5.1 Conferences and Publications

Throughout the course of this study, the author had the opportunity to present and demonstrate the project at several conferences as well as publication at ICMC 2012.

- **Leeds International Festival for Innovations in Music Production and Composition (iFIMPaC), April 2012.** This is an annual event held at the Leeds College of Music. iFIMPaC creates a unique environment for composers, producers, music industry representatives, academics, educators and students to discuss their compositional, pedagogical and production work as practice-led research. The author was invited for a presentation and demonstration (Leeds College of Music, 2012).
- **Sonic Arts Forum, July 2012, Leeds.** This seeks to provide opportunities for those involved in the sonic arts to meet, exchange ideas, present new work, develop new tools, discuss aesthetic issues and investigate practice. The author was invited in July 2012 for a presentation and demonstration.
- **The International Computer Music Conference (ICMC), September 2012, Ljubeljana, Slovenia.** Since 1974 the International Computer Music Conference has been the major international forum for the presentation of the full range of outcomes from technical and musical research, both musical and theoretical, related to the use of computers in music. This annual conference regularly travels the globe, with recent conferences in the Americas, Europe and Asia. The author was featured in the conference publication (Jidkov et al, 2012) and presented a poster in Ljubeljana.
- **Sound on Sound Magazine, January 2013.** The author is a member of the Adaptive Music Technology Research group, University of Huddersfield. This group was profiled for an article in the January 2013 edition of the magazine (Thomas, 2013). The study application was mentioned in the article.

5.2 ICMC 2012 PUBLICATION

_ICMC2012 NON-COCHLEAR SOUND LJUBLJANA _9.–14. SEPTEMBER

A WIRELESS, REAL-TIME CONTROLLER FOR THE ENSEMBLE AUDIO-VISUAL SYSTEM.

Anton Jidkov
Music Technology
Computing and Engineering
University of Huddersfield
anton.jidkov@gmail.com

Dr Ian Gibson
Music Technology
Computing and Engineering
University of Huddersfield
i.s.gibson@hud.ac.uk

Mark Hildred
Apollo Creative
mark@apollocreative.co.uk

ABSTRACT

This research investigates development of sound diffusion software for the Apple iPhone and iPad. It is specifically written for Apollo Creative's 'Ensemble' system, which is an interactive audio, lighting and video software and hardware package. It allows users to design software instruments for sound installations and shows, reacting to a range of input sensors. The iPhone software controls the surround sound parameters of Ensemble. The user interface is designed to accommodate a range of users, including educators, composers and sound designers. The software incorporates the OpenFrameworks library [9] and communicates with 'Ensemble' using the Open Sound Control (OSC) [15] protocol over wireless TCP/IP.

1. THE ENSEMBLE SYSTEM

The Apollo Ensemble [1] is a system designed for teachers and special needs specialists, allowing them to configure interactive sensory environments for individuals with a range of disabilities. Ensemble also has applications in the areas of exhibitions, artistic installations and children's play areas.

Ensemble is a switch operated system that is an evolution of Midicreator [7]. Switch operated hardware allows people with disabilities to have a range of control methods that can be mapped to output stimuli via specialised software. Similar examples include the Skoog [11], a tactile music instrument; MidiMate [5], an access device for electronic keyboards and Quintet [13].

The Ensemble system is split into three main components: a range of input sensors for detecting movements; designer and player software running on a PC; and output devices which can include sound, lighting, video, image and sensory equipment.

1.1 The Hub interface

The Ensemble Hub, figure 1, forms the main USB interface for the PC, featuring four sockets for simple on/off switches and two for variable sensors. It also contains a 433MHz transmitter for controlling proprietary sensory equipment and a 2.4GHz module for wireless sensors. A basic portable setup can be achieved by using the Hub together with a laptop or netbook PC.

Figure 1. Ensemble Hub

1.2 Input sensors

Due to the nature of the special needs market, there are a large number of standard 'assistive technology' switches available for a wide variety of needs. Past system development has focused on wireless adaptors. It is important that there is a clear link between cause (i.e. sensors employed) and effect (sonic output) in special needs work [2]. Non-contact sensors can be problematic in these environments. Also sensors need to be robust and simple to use.

The current range of sensors include:

- Connect – adaptor to allow up to four switches to be linked wirelessly to the Ensemble Hub.
- Dice – each side of the sensor can be used to trigger a different sound or effect.

Figure 2. Players using the dice sensor within a foam cube.

- Press – sensitive pressure pad that produces a variable signal depending on the amount of pressure applied.
- Tilt – produces a variable pitch and roll output as it is rotated.
- Squeeze – an air pressure bulb whose output varies with the pressure applied.
- Dual – adaptor to allow up to two variable sensors to be linked wirelessly to the Ensemble Hub.

1.3 Designer

The Designer software is a drag-and-drop environment, with blocks onscreen representing the key components of the system shown in figure 3. These are linked together to provide a 'map' of how the system operates.

Each block can have a number of settings or options, which are adjusted in a panel on the right hand side of the screen. Where a greater number of options are available, a separate editor window can be used, for example when entering MIDI notes or adjusting a colour on a light.

As well as input and output blocks, the software allows for tools, which can alter the behaviour of a signal in the system. These allow the user to add delays, expand the range of a signal or alter how a switch operates.

The Designer software is unaware of exactly what devices it is using, instead relying on rules for connecting and setting-up the blocks. All blocks are defined by an xml file, which can be altered to produce blocks specific to a user's installation.

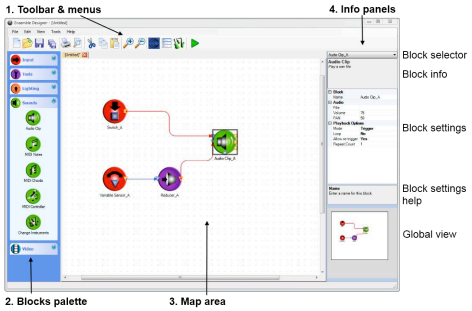


Figure 3. The Designer software.

1.4 Player

The Player application, shown in figure 4, separates the actual 'playing' of maps away from the Designer, allowing finished maps to be played without the need for the Designer to be open. This presents a friendly, CD-player style interface for users to interact with. It is particularly important in schools where any hint at complexity can stop people from using the software.

Figure 4. The Player.

1.5 Devices Monitor

Services are handled by a tray application, which updates dynamically as devices are connected and removed from the system. Separate services interpret messages for the Player software and deal directly with communication with the hardware. In this way, new services can be developed to add additional functionality to the software.

Current services include:

- DMX Output [14] – for lighting control.
- MIDI Output [8].
- Gamepad Input – allows any device that appears to the PC as a gamepad to be used with the Ensemble.

2. IPHONE/IPAD APPLICATION

2.1. Overview

The mobile control module will primarily aim to satisfy the needs of teachers and special needs specialists. This requires the application to have simple controls affecting a wide range of parameters and for the controls to be intuitive and familiar so that the application can be understood and used easily. The application will also provide features for other user groups such as composers and sound designers who are able to invest more time in exploring the system.

The application controls Ensemble's surround sound parameters. The interface is created dynamically by requesting the sound outputs and external inputs of the user design 'map' from Ensemble. A corresponding design map is recreated on the iPhone/iPad interface.

The design was implemented for the iOS operating system primarily using Openframeworks [9]. This is an open source C++ toolkit that was used within the Xcode integrated development environment (IDE).

There are several existing applications that allow users to build dynamic controllers currently available for iOS including TouchOSC [4], Liine Lemur [6] and Fantastick [10]. However these systems do not specialise in surround sound, provide limited automation and path manipulation.

2.2. Design

2.2.1. Routing of Sound

Initially, sound outputs and listener position are displayed on-screen in a grid. These icons can be arranged to reflect the position of speakers and the listener in the physical environment. Figure 5 shows a 5 speaker arrangement and the listener positioned by the user. Speaker and listener icons can be positioned using the multi-touch capabilities of the device. The user is able to navigate and zoom around the display area as well as hide the toolbars for improved design control.

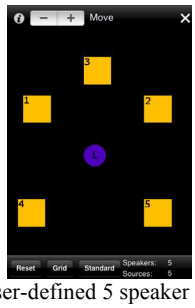


Figure 5. User-defined 5 speaker arrangement.

2.2.2. *Main Volume Control*

The volume of the speakers can be controlled individually as well as globally. The user can make a custom selection of speakers or choose a selection option: 'Left', 'Right', 'Centre' and 'All'. These options select speakers depending on their position defined by the user.

Volume control has 2 modes: Absolute and Relative. In Absolute mode, the volume of all speakers is changed simultaneously. In Relative mode, as speaker volume is increased or decreased, the relative volume differences between speakers are retained until all the speakers are pushed to minimum or maximum volume.

Figure 6 shows the 'Right' speakers selected in Absolute mode. Volume is changed using the fader in the lower menu.

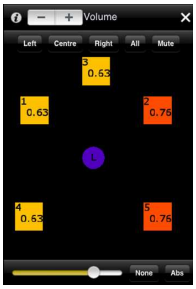


Figure 6. Volume selection and modification.

2.2.3. *Dynamic Sound Trajectory Control*

The prototype application has 3 sound trajectories (A, B, C), each with 2 pan layers, which the user can configure, control and edit. In pan layer 1, a path is drawn directly onto the speaker array display. Once completed, the path is followed at the same speed as it was drawn.

The direction of playback can be reversed and the speed can be changed. In 'Single Play' mode, the loop plays once, following the pre-defined path, stopping when it is complete.

Sound sources can be routed to one of the sound trajectories. Multiple sources can be routed to a single trajectory. Figure 7 shows 3 different trajectories displayed on the speaker array. The top menu allows editing of the selected trajectory. The bottom menu shows the 3 trajectories visible and the routing of the input sources. They are colour coded for quick reference during a performance.

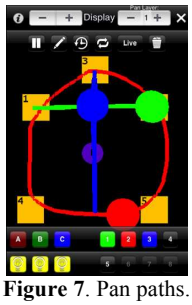


Figure 7. Pan paths.

The path can be manipulated as it is playing using touch gestures. The path can be moved around the screen, resized and rotated as shown with the shape in Figure 8.

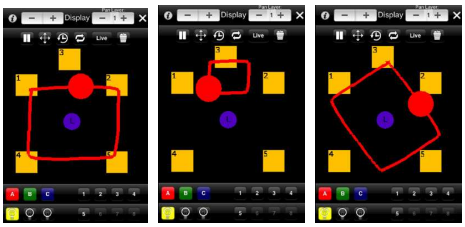


Figure 8. Touch manipulation.

The sound trajectory control can be set to 'Live Mode' allowing the path to be directly controlled by the user. Figure 9 shows trajectory C displayed in Live mode. The path is drawn directly onto the screen and leaves a short tail to show previous positions.

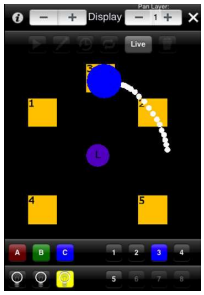


Figure 9. Live pan.

After the user has created an initial sound trajectory, it can be automated further by creating a second pan layer that moves the first layer. Layer 2 can be controlled and manipulated in the same way as layer 1 (move, rotate and resize). Figure 10 shows a horizontal stereo pan, layer 1, being moved towards the rear speakers by the layer 2 path.

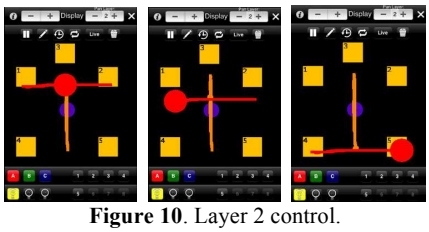


Figure 10. Layer 2 control.

2.2.4. User Interface

The User Interface (UI) design is important to appeal to a variety of users. Whereas sound designers, familiar with technical control of audio, may allow for complex control at the expense of the UI, a large proportion of the users (educators) will require a more immediately accessible control interface. Therefore the application will manage workflow and available options depending on the users' required level of control and proficiency employing a habituation pattern for menu navigation and an incremental construction pattern of the speaker array and sound trajectory design [12].

3. OSC IMPLEMENTATION

OSC messages are sent between the Ensemble system and the mobile device. OSC was developed at CNMAT primarily as a protocol for communication between computers, sound synthesizers, and other multimedia devices optimized for modern networking technology [15].

3.1. Protocol Specification

OSC employs an 'open-ended URL-style symbolic naming scheme'. The project aims to carry out most of the data processing within the device and to send out as few different messages as possible.

3.1.1. Configuration Messages

At the setup stage, the application sends a request message ('/getconfig') to obtain the quantity of speaker outputs and user inputs. The return messages are in the following form. For the speaker outputs: '/config/speakerquantity s'. Where s = the amount of speakers in the design. For the user inputs: '/config/soundsources i1'. Where i1= the amount of input sources controlling sound.

3.1.2. Control Messages

During operation, messages are sent when the main volume is changed. The message consists of the speaker number and the master volume in the form: '/speaker/vol n v1'. Where n = the speaker number; v1= master volume (0-1).

Messages are sent when the input sources are routed to playing sound trajectories or a changing live mode. Messages are sent in the form: '/speaker/sound n i2 v2'. Where n = the speaker number; i2 = the sound source number; v2 = the percentage scale of the speaker volume (0-1). The percentage scale value is determined by using distance-dependent panning principles utilizing the x and y speaker distances.

Additional OSC messages can be sent which specify the x and y distance between sound source and speaker to allow for easier interfacing with the Ensemble environment.

4. TESTING AND FURTHER WORK

The controller application has been tested successfully with a range of surround sound setups using an emulator designed in Max MSP [3]. Apollo Creative has provided user interface and gesture control feedback. Further user testing is to follow.

Further features currently being designed include advanced time manipulation and display; multi-touch control for additional parameters and advanced path manipulation.

5. REFERENCES

[1] Apollo Ensemble. [online] <www.apolloensemble.co.uk>, 2011.

[2] Corke, M. *Approaches to Communication through Music,* David Fulton, London, 2002.

[3] Cycling '74. Max MSP. [online] <cycling74.com/products/max>, 2012.

[4] Hexler. [online] <hexler.net/software/touchosc>, 2012.

[5] Knox, R. "*Adapted Music As Community Music",* International Journal of Community Music, Vol. 1, 2004.

[6] Liine. [online] <liine.net/en/products/lemur>, 2012.

[7] MidiCreator. [online] <www.midicreator-resources.co.uk/index.php>, 2012.

[8] MIDI Manufaturers Association. [online] <www.midi.org/aboutmidi/index.php>, 2012.

[9] OpenFrameworks. [online] <www.openframeworks.cc/about>, 2012.

[10] Pink Twins. [online] <pinktwins.com/fantastick>, 2012.

[11] Skoogmusic Ltd. [online] <www.skoogmusic.com>, 2012.

[12] Tidwell, J. *Designing Interfaces*, O'Reilly, USA, 2005.

[13] Unique Perspectives Ltd. [online] <www.click2go.ie/all-products/quintet>, 2012.

[14] USITT. DMX512 [online] <www.usitt.org/Resources/Standards2/DMX512/DMX512FAQ>, 2012.

[15] Wright, M. *Open Sound Control: an enabling technology for musical networking*, Cambridge University Press, UK, 2005.

5.3 TEST RESULTS

5.3.1 SPEAKER MOVEMENT

Speakers moving	Distance	Speed (s)	Total Load (%) - 1	Total Load (%) - 2	Total Load (%) - 3	Average (%)
1	Horizontal	1	49.5	50.9	49.5	49.96666667
	Horizontal	5	54.6	56.5	55	55.36666667
	Vertical	1	51.4	52.7	55.1	53.06666667
	Vertical	5	55.6	55.6	54.6	55.26666667
2	Horizontal	1	48.1	51.9	52.3	50.76666667
	Horizontal	5	57.8	57.8	57.4	57.66666667
	Vertical	1	56.6	60.6	55	57.4
	Vertical	5	56.5	57.8	56.1	56.8
3	Horizontal	1	56.4	55	55.1	55.5
	Horizontal	5	56.9	57.8	57.8	57.5
	Vertical	1	53.6	56	55.6	55.06666667
	Vertical	5	57.4	58.3	58.7	58.13333333
4	Horizontal	1	55.1	55	57.9	56
	Horizontal	5	60	58.2	59.3	59.16666667
	Vertical	1	57	57.4	50	54.8
	Vertical	5	62	59.1	61.5	60.86666667
5	Horizontal	1	61.1	56.4	52.7	56.73333333
	Horizontal	5	60	60	61.5	60.5
	Vertical	1	58.3	59.6	56.5	58.13333333
	Vertical	5	61.8	60.6	60.2	60.86666667
Whole Screen Translate	Horizontal	1	49.1	53.2	52.3	51.53333333
	Horizontal	5	54.6	54.6	56	55.06666667
	Vertical	1	49.1	51.8	51.9	50.93333333
	Vertical	5	54.6	55.6	57	55.73333333

5.3.2 VOLUME

Speakers	Speed (s) / mute	Total Load (%)
1	1	79.6
	5	71.2
	Mute	48.6
2	1	65.7
	5	73.9
	Mute	48.6
3	1	61.7
	5	70.6
	Mute	49.1
4	1	64.5
	5	70.5
	Mute	48.1
5	1	58.3
	5	72.4
	Mute	49.5

5.3.3 LIVE MODE

Movement	Routing quantity	Speed (s)	Total Load (%)
Horizontal	1	1	77.3
	4	1	77.4
	8	1	88.6
	1	5	81.8
	4	5	84.1
	8	5	89.4
Vertical	1	1	77
	4	1	80.9
	8	1	84.6
	1	5	78.8
	4	5	82.8
	8	5	89.6

5.3.4 PLAYBACK

Paths playing	Array size	Speed (S)	Routing (sources)	Total Load (%)
1	Small	1	1	51.4
		1	4	58.7
		1	8	66.4
		10	1	55.6
		10	4	61.8
		10	8	68.5
	Large	1	1	76.8
		1	4	81.7
		1	8	93.4
		10	1	93.5
		10	4	88.8
		10	8	89.1
2	Small	1	1	59.4
		1	4	62.4
		1	8	70.3
		10	1	64.5
		10	4	67.9
		10	8	76.3
	Large	1	1	94.3
		1	4	93.3
		1	8	91.5
		10	1	76.6
		10	4	71.8
		10	8	66.8
3	Small	1	1	62.7
		1	4	70.5
		1	8	76.8
		10	1	68.8
		10	4	75.2
		10	8	82.6
	Large	1	1	64.8
		1	4	69.7
		1	8	91.6
		10	1	69.1
		10	4	72.6
		10	8	76.6
4	Small	1	1	78.5
		1	4	83
		1	8	88.3
		10	1	92
		10	4	95.6
		10	8	73.6
	Large	1	1	90.1

		1	4	91.1
		1	8	92.4
		10	1	94.7
		10	4	95.3
		10	8	90.8
5	Small	1	1	71.8
		1	4	63.6
		1	8	68.2
		10	1	67.9
		10	4	68.5
		10	8	71.7
	Large	1	1	89.2
		1	4	91.2
		1	8	95.3
		10	1	84.8
		10	4	85.2
		10	8	87.7
6	Small	1	1	69.2
		1	4	72.7
		1	8	76.8
		10	1	82.8
		10	4	82.9
		10	8	85.2
	Large	1	1	94.3
		1	4	95.3
		1	8	96.4
		10	1	88.7
		10	4	89.9
		10	8	92

5.3.5 MESSAGE TIMING

Path Size	**Paths playing**	**Routing (sources)**	**Time (mS)**
Small	1	1	827.1
	1	8	856.29
	3	1	827.06
	3	8	835.29
	6	1	825.78
	6	8	853.8
Large	1	1	8309.7
	1	8	8219.24
	3	1	8319.5
	3	8	8221.71
	6	1	8152
	6	8	8196.37

5.3.6 MOVE

Array Size	Movement	Speed of gesture (S)	Total Load (%)
Small	Horizontal	1	59.3
		5	63.3
	Vertical	1	56.9
		5	66.4
Large	Horizontal	1	87.3
		5	89.3
	Vertical	1	89.7
		5	92.6

5.3.7 RESIZE

Array Size	Speed of gesture (S)	Total Load (%)
Small	1	63.3
	5	60.6
Large	1	92
	5	86.6

5.3.8 ROTATE

Array Size	Resize amount	Speed of gesture (S)	Total Load (%)
Small	90 degrees	1	59.3
		5	63.3
	360 degrees	1	71.4
		5	71.6
Large	90 degrees	1	90.9
		5	88.8
	360 degrees	1	96.2
		5	94.5

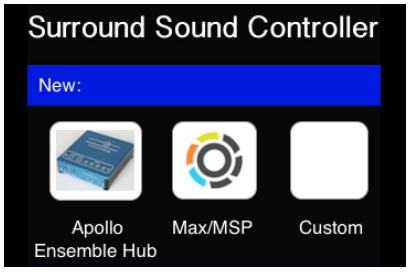
5.4 USER MANUAL

CONTENTS

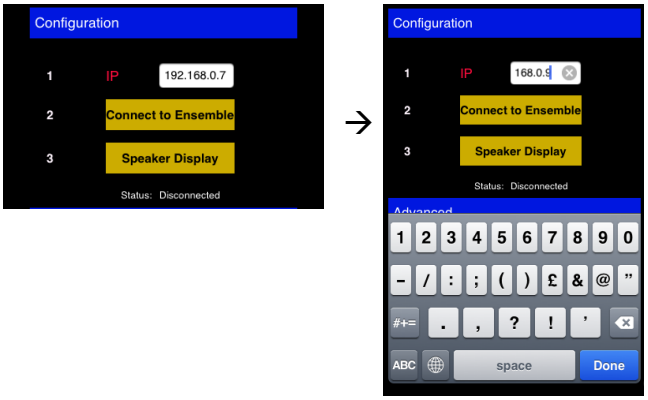
5.4.1 Setting up a New Project	144
5.4.2 Finding the IP of your Computer and iOS Device.	144
5.4.3 Setting up an Ad-Hoc Connection	145
5.4.4 Positioning the Speakers and Listener	145
5.4.5 Showing and Hiding Toolbars	146
5.4.6 Navigating Modes	146
5.4.7 Volume Mode	146
5.4.8 Display Mode	147
5.4.8.1 Selecting a Pan Path and Routing Sound Sources	147
5.4.8.2 Live Mode	147
5.4.8.3 Drawing a Path	148
5.4.8.4 Playing a Path	148
5.4.8.5 Continuous/Single Play	148
5.4.8.6 Reverse a Path	148
5.4.8.7 Change the Speed of a Path	148
5.4.8.8 Using Touch Gestures	149
5.4.8.9 Using Multiple Layers	149
5.4.9 Saving a Project	151
5.4.10 Loading a Project	151
5.4.11 Time Indicators	152

5.4.1 Setting up a New Project

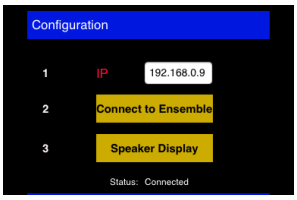
1. Start the app.
2. Press the 'Apollo Ensemble Hub' under the 'New' title.



3. In the configuration page, set the IP of the computer controlling the panning. This page can be accessed through the options menu.



4. Press 'Connect to Ensemble'. If successful, the 'status' will read: Connected.
5. Press 'Speaker Display'.



5.4.2 Finding the IP of your Computer and iOS Device

Mac OSX:

1. Go to System Preferences > Network.
2. The IP address is displayed under Status.

Windows 7:

1. Go to Control panel > Network and Sharing Centre.
2. Click Network Connection Status.
3. Click Details. The IP is shown as the IPv4 address.

iOS device:

1. Go to Settings > Wi-Fi.
2. Tap the blue arrow next to the network that you are using.
3. The IP address is displayed.

5.4.3 Setting up an Ad-Hoc Connection

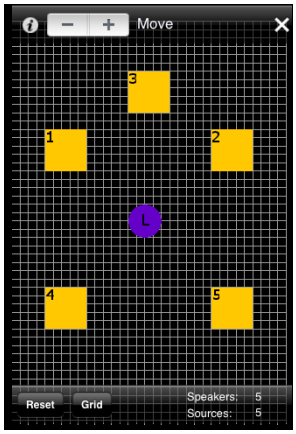
Mac OSX:

1. Go to System Preferences > Network.
2. Open the drop down menu under Network Name.
3. Choose Create Network.
4. Select a name and a password for your network.
5. Connect your iOS device normally.

Windows 7:

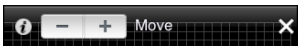
1. Click the Start button > Connect to Network.
2. Click Set up a connection or network.
3. Click Set up an ad hoc (computer-to-computer) network. Follow the setup wizard.
4. Connect your iOS device normally.

5.4.4 Positioning the Speakers and Listener



- When you have set up a new project, the speakers are displayed in a grid. The listener position is displayed as a purple circle.
- Position objects by touching and sliding them around the screen.
- The entire display can be navigated by touching outside the speakers and listener.
- The listener position can be hidden/displayed via the options menu. This can be used in situations where the listener is not fixed (e.g. an installation) or undefined.
- The grid button enables a grid that will snap objects in place and help alignment.
- The reset button repositions objects to their original positions.

5.4.5 Showing and Hiding Toolbars



- The toolbars can be made visible by pressing the 'i' button in the top right corner of the screen.
- The toolbars can be hidden by pressing the 'X' button the top right corner of the screen.

5.4.6 Navigating Modes

There are 3 modes of operation:

'Move' – To move the speakers

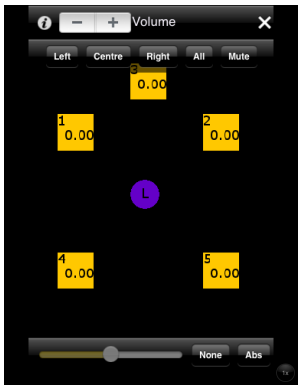
'Volume' – To set the volume.

'Display' – To move the sound and to draw/play/manipulate pan paths.

To navigate around these modes, use the 'stepper' on the top left of the screen.

5.4.7 Volume Mode

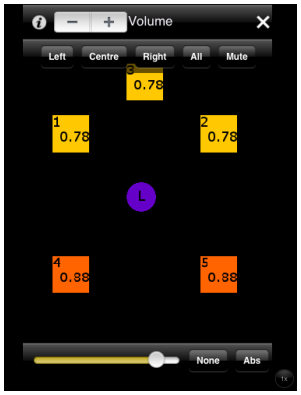
Volume is changed by selecting speakers and using the slider.



The speakers can be selected in several ways:

- Touching the speakers.
- Using the buttons on the top toolbar – location based speaker group selection.

The speakers can be muted and unmuted using the 'mute' button.



For multiple speakers the volume can be changed in absolute and relative mode by tapping the button on the bottom right.

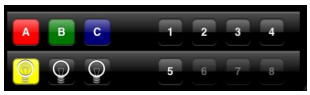
- In Absolute mode, all the selected speakers become the same volume when changing the value.
- Relative mode retains differences between the speaker volumes when changing the value. If the slider goes too high or low, volumes will all be set to 0 or 1.

5.4.8 Display Mode

5.4.8.1 Selecting a Pan Path and Routing Sound Sources

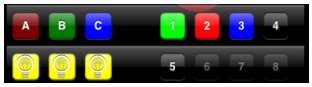
The bottom toolbar allows you to select one of the 3 pan paths to operate (A, B & C). These paths can be selected/deselected by tapping on them. Paths can be made visible/invisible by tapping the 'light bulb' icon beneath each one.

Here, only pan path A is selected and visible:



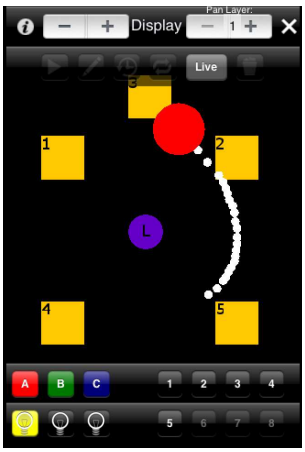
Paths are routed to 'sound sources' (e.g. motion sensors mapped to audio with Ensemble) using the routing buttons on the right. Up to 8 sources are available. To route/unroute sources, select a pan path and tap the sound source. The sound source will be coloured corresponding to its routing.

Here, source 1 is routed to pan path B (green), source 2 is routed to pan path A (red) and source 3 is routed to pan path C (blue). Sources 4 & 5 are unrouted:



5.4.8.2 Live Mode

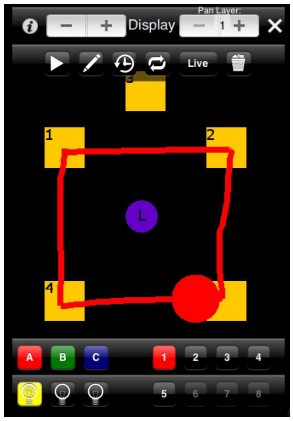
Live mode is used to position the sound directly.



1. Select a pan path and route the required sound sources using the bottom toolbar.
2. Tap the 'Live' button in the top toolbar.
3. Position sound with your finger. The sound position is shown as a circle.

5.4.8.3 Drawing a Path

Positioning the sound can be automated by drawing a path.



- Select a pan path and route required sources.
- Tap the second button from the left (top toolbar) until it is displaying the pencil icon (draw mode):
- Draw a path directly onto the speakers in the way that you would like it to play back.

The diagram on the right shows pan trajectory A selected, sound source 1 routed and a path drawn. The sound position is shown as a red circle.

To redraw a path either:

- Overwrite the current path by drawing over it.
- Or, press the 'delete' button on the top toolbar:

5.4.8.4 Playing a Path

After drawing the path, it can be played back. Select the trajectory you want to play from the bottom toolbar. Press the play button (top toolbar, far left). Press it again to pause the path.

5.4.8.5 Continuous/Single Play

By default, the path will play back continuously. To make the path play once, select the 'single play' button on the top toolbar:

This will stop the sound moving when it reaches the end of the path.

5.4.8.6 Reverse a Path

The direction of travel of the path can be reversed by pressing the 'reverse' button:

5.4.8.7 Change the Speed of a Path

The speed of the path playback can be changed. Select the path that you want to manipulate from the bottom toolbar. Select the 'speed' mode on the top toolbar:

Move a finger up the screen to increase the speed and down the screen to decrease the speed of playback.

5.4.8.8 Using Touch Gestures

Once the path has been drawn, it can be manipulated using touch gestures.

1. Select the path you want to manipulate from the bottom toolbar
2. Continuously press the second from the left button on the top toolbar to activate touch gesture mode:

As the path is playing, three gestures can be used to move, resize and rotate the path.

Moving

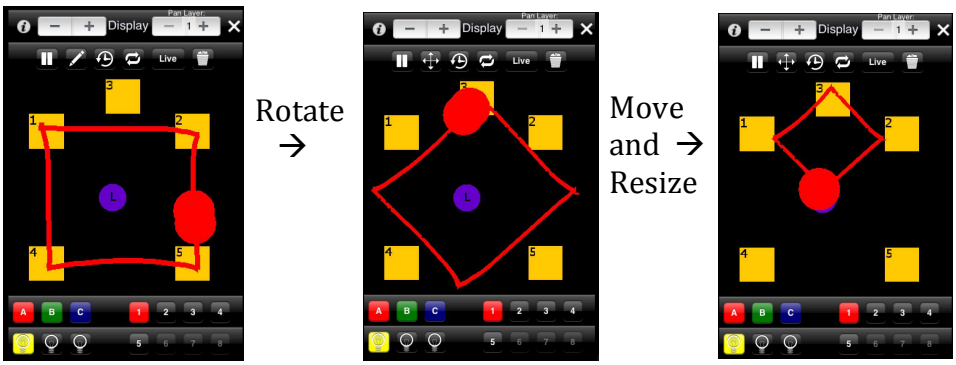
To move the path, simply use **one finger** to move the path to the desired location.

Resizing

To resize the path, using **two fingers,** pinch in or pinch out of the path to resize it

Rotating

Use **three fingers** to rotate the path.



5.4.8.9 Using Multiple Layers

The process of moving the layer using touch gestures can be automated. This is done using the multi-layer functionality of the app.

2 layers are available:

Layer 1: This is the primary layer where you can draw a path

Layer 2: This is the secondary layer that moves layer 1.

To select a layer, use the stepper on the top right of the screen:

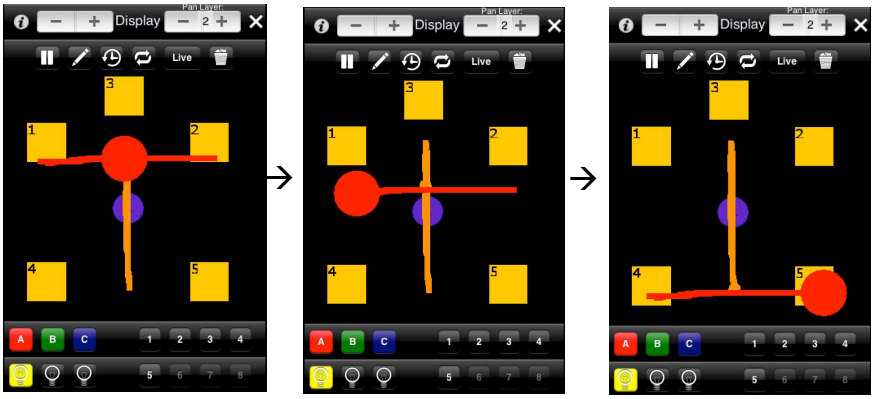
To draw a second layer:

1. Select the path trajectory from the bottom toolbar.
2. Make sure you have drawn a layer 1 path.
3. Select layer 2 using the stepper.
4. Draw a path as normal.

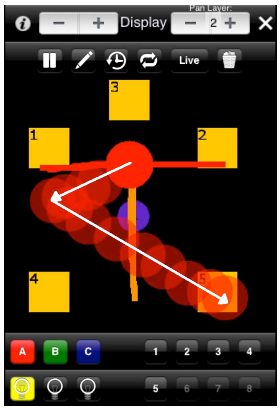
The same manipulation controls that were available for layer 1 are available for layer 2. Therefore you can,

- Reverse
- Single play
- Change speed
- Use touch gestures

The diagrams below show an example of a layer 1 horizontal path (red) being moved by a layer 2 (orange) path towards speakers 4 and 5.

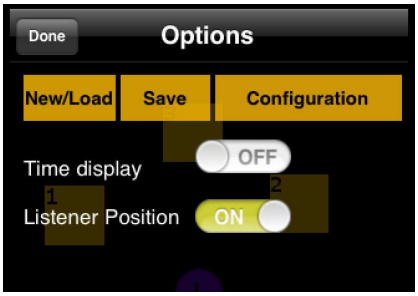


The overall movement of the sound is shown below:



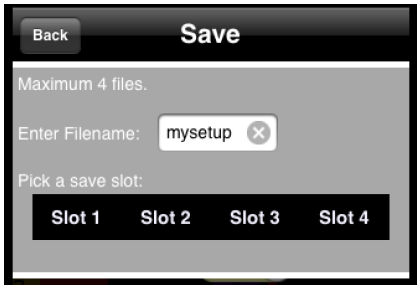
5.4.9 Saving a Project

You can save the project settings including:



- Speaker positions
- Speaker volumes
- Routing
- Paths

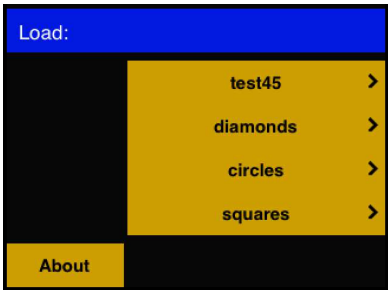
To save a project:



1. Go to the options menu by pressing the 'i' button on the top left of the screen.
2. Press the 'Save' button:
3. This opens a dialog box.
4. You can save up to 4 files.
5. First enter a filename.
6. Select a save slot.
7. 'Saved!' will be display below the slot numbers.

5.4.10 Loading a Project

Loading a project can be done from the New/Load Screen. This is shown when the app is started initially. It can also be accessed from the options menu by pressing the new/load button.

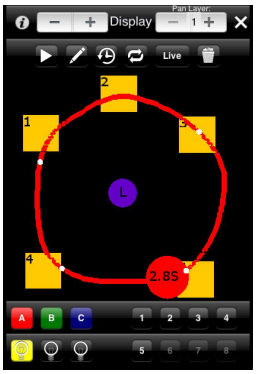


From the new load screen, 4 slots are displayed with custom file names.

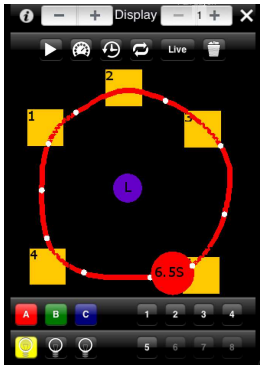
Tapping one of the projects, loads it.

5.4.11 TIME INDICATORS

Time indicator can be displayed on the layer 1 path. These show second boundaries and display the path time within the sound source. These can be switched on/off in the options menu by operating the time display switch. The images below show the same path, with time indicators, at two different speeds.



Fast



Slow